MSX

truuks en tips
deel 2

A.C.J. Groeneveld



MSX truuks en tips deel 2

A.C.J. Groeneveld



uitgeverij STARK - TEXEL
postbus 302 - 1794 ZG Oosterend tel. 02223 - 661

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Groeneveld, A.C.J.

MSX truuks en tips / A.C.J. Groeneveld.—Oosterend: Stark-Texel
Dl. 2
ISBN 90-6398-340-9
SISO 365.3 UDC 681.3.06
Trefw.: programmeren (computer)/MSX (computer).

. . . .

1e druk 1985
ISBN 90 6398 340 9

Voorwoord

In het eerste deel van MSX TRUUKS en TIPS behandelden we veel korte maar vaak zo handige programma's of programma-onderdelen.

Dit tweede deel bevat over het algemeen meer wat langere programma's die vaak een afrondend geheel vormen.

In dit tweede deel gaan we, binnen de BASIC-taal, ondermeer op ontdekkingsreis binnen onze MSX-computer. We leren de Programmable Sound Generator kennen, gaan op strooptocht in het video-RAM, keren een basic-programma binnenste-buiten en brengen wat beveiligingen op onze programmatuur aan.

Ook leren we twee nieuwe MSX-kommando's kennen. Kommando's die in bijna geen enkel boek worden besproken en die daarom door maar weinigen worden beheerst.

De floppy-bezitter krijgt in dit boek de mogelijkheid om de inhoud van zijn complete floppy-disk te onderzoeken, buiten de MSX-struktuur om. Hij kan zijn floppies ook een serienummer geven dat niet kopiëerbaar is en waarmee hij foute plaatsing van floppies kan voorkomen of programma's mee kan beveiligen.

Kortom, voor de rechtgeaarde amateur is dit tweede deel misschien soms wat moeilijk maar in ieder geval razend interessant.

Terwijl u dit tweede deel doorworstelt, begin ik vast aan deel drie.

november 1985,
A.C.J. Groeneveld.

INHOUD

1 Het binaire stelsel

In de volgende hoofdstukken gaan we wat dieper in op de Programmable Sound Generator en de Video Display Processor; twee kleine computertjes binnen de MSX-standaard die samen zorgen voor ondermeer het beeld en geluid van de MSX-computer.

Het is erg interessant om binnen MSX wat meer met dit duo te ondernemen. Helaas is hiervoor een beetje kennis nodig van talstelsels; met name van het binaire stelsel.

Het talstelsel waarin wij normaal werken, is het tientallig stelsel. Dit stelsel heet zo omdat we binnen dit stelsel gebruik maken van tienvouden. Om een getal met een waarde van tien of hoger te kunnen noteren, moeten we in ons talstelsel gebruik maken van twee cijfers. Op dat moment hebben niet alleen de cijfers een waarde maar worden er ook waarden toegekend aan de posities waarop die cijfers worden vermeld. Bijvoorbeeld het getal 1623:

```
   1            6            2    3
1 maal 1000 + 6 maal 100 + 2 maal 10 + 3  = 1623
```

Het cijfer 1 in dit voorbeeld is niet één maar duizend maal één waard. Dit komt omdat dit cijfer 1 op de vierde positie staat. Omdat het cijfer 2 op de tweede positie staat, is het twee maal tien is twintig waard.

Elke computer werkt op één of andere manier met het tweetallige stelsel. In het tweetallige stelsel is een cijfer wanneer het een positie verder naar links staat niet tien maal maar slechts twee maal zoveel waard. Een voorbeeld:

```
  1          1        1
1 maal 4 + 1 maal 2 + 1  = 7
```

Het binaire getal 111 heeft voor ons dus de waarde 7.

In het binaire stelsel is een cijfer steeds twee maal zoveel waard wanneer het een positie verder naar links komt te staan. Hierdoor is het niet nodig om cijfers hoger dan het cijfer 1 te gebruiken. Immers, het getal 2 kunnen we binair eenvoudig schrijven als één maal twee, dus 10.

Een binair getal omrekenen naar een decimaal getal is met deze kennis eenvoudig. Enkele voorbeelden:

1111	1x8+1x4+1x2+1=15
11110	1x16+1x8+1x4+1x2=30
1001	1x8+1=9
11111111	1x128+1x64+1x32+1x16+1x8+1x4+1x2+1=255

Een decimaal getal omrekenen naar een binair getal is wat moeilijker. Er is een ezelsbruggetje voor dat we hier in een voorbeeld laten zien:

Bijvoorbeeld het (decimale) getal 205

205	is een oneven getal (niet deelbaar door twee) omdat het oneven is, noteren we het cijfer 1 op de laatste plaats. Vervolgens delen we dit getal door 2 en verwaarlozen we het cijfer na de komma. 205/2=102.5. We noteren het getal 102.	1
102	is een even getal. Daarom noteren we een cijfer 0 voor het eerder geplaatste cijfer en we delen het getal weer door 2. 102/2=51.	01
51	is oneven. Het cijfer 1 noteren, 51 door 2 delen en het cijfer na de komma verwaarlozen.	101
25	is weer oneven. Het cijfer 1 noteren en door 2 delen.	1101
12	is even. We noteren een 0 en delen door 2.	01101
6	alweer even...	001101
3	oneven.	1001101

1 oneven. 11001101

Wanneer we nu het getal 1 door twee gaan delen, krijgen we de uitkomst 0.5. Bij verwaarlozing van de cijfers na de komma blijft er nul over. Dit betekent dat we klaar zijn met rekenen. Konklusie:

11001101 binair is gelijk aan 205 decimaal.

Een algemene beschrijving (de vakman noemt het een ALGORITHME) van deze methode:

Een tientallig (decimaal) getal kan men als volgt omrekenen naar een tweetallig (binair) getal: steeds kijken we eerst of een getal even is. Zo ja, noteren we het cijfer 1, zo nee, noteren we het cijfer 0. Vervolgens delen we het decimale getal door twee en verwaarlozen we de cijfers na de komma. Tenzij het resultaat op nul uitkomt, herhalen we ons recept. De enen en nullen noteren we bij deze methode van rechts naar links.

De MSX-computer biedt de mogelijkheid om de berekening van binair naar decimaal en van decimaal naar binair door het systeem te laten doen. Een voorbeeld: we rekenen het getal 205 naar het binaire stelsel om. Onder BASIC tikken we in:

```
PRINT BIN$(205)
```

De computer antwoordt onmiddellijk met:

```
11001101
Ok
```

Wanneer we de berekening net andersom willen laten uitvoeren, tikken we bijvoorbeeld in:

```
PRINT &B11001101
```

De computer antwoordt onmiddellijk met:

```
 205
Ok
```

Konklusie: binnen MSX-basic kunnen we met de BIN$()-funktie van een decimaal getal een binair getal maken. Door een getal

te voorzien van het voorvoegsel &B geeft men te kennen dat het om een binair getal gaat; de computer kan dit getal naar het decimale stelsel terugrekenen.

Nu we het binaire stelsel op deze manier een beetje beheersen, kunnen we ons wat in de diepere kelders van het MSX-basic begeven. In het volgende hoofdstuk gaan we in op de wederwaardigheden van de Programmable Sound Generator ofwel de PSG. Dit computertje-in-een-computer regelt al ons MSX-geluid en heeft veel meer mogelijkheden dan u op het eerste gezicht vermoedt...

2 De programmable sound generator

De PSG, zoals we de Programmable Sound Generator verder zullen noemen, bestaat in feite uit één enkele chip. Wanneer u met het (u waarschijnlijk bekende) PLAY-kommando stukjes muziek programmeert, dan geeft u in feite het MSX-basic de opdracht om deze PSG te gaan gebruiken.

We kunnen de PSG echter ook 'een beetje buiten MSX-basic om' gebruiken en wel via het SOUND-kommando. Dit MSX-kommando staat wat dichter bij de computer en wat verder weg van de gebruiker. Daarom is het gebruik van het SOUND-kommando ook niet gemakkelijk.

Een voordeel is het dat er plotseling veel meer mogelijkheden blijken te zijn wanneer men gebruik maakt van het SOUND-kommando.

Eén tip vooraf: bij het spelen met het SOUND-kommando gaat er wel eens iets mis. Dit is niet gevaarlijk voor de computer; het mag best wel eens een keer fout lopen. Vervelend is het echter dat het voor kan komen dat we door een foutje de PSG hebben aangezet en hem niet meer hebben uitgezet. We moeten óf het volume dicht draaien óf blijven luisteren naar het geluid dat de PSG tijdens het werken in basic voortdurend maakt. Echter, de PSG kan volledig worden ge'reset' door één keer een CONTROL-STOP in te geven. Het vervelende, mislukte geluid is dan onmiddellijk verdwenen.

De PSG wordt bestuurd door veertien zogenaamde REGISTERS. Deze registers kunnen allemaal een waarde bevatten. De waarden van alle registers samen bepalen, wat de PSG doet of gaat doen. Deze registers kunnen we met behulp van het SOUND-kommando van waarden voorzien.

De PSG-registers zijn als volgt opgebouwd:

DE PSG REGISTERS

register										
0									toonhoogte kanaal A	→ sound 0,...
1										→ sound 1,...
2									toonhoogte kanaal B	→ sound 2,...
3										→ sound 3,...
4									toonhoogte kanaal C	→ sound 4,...
5										→ sound 5,...
6									toonhoogte ruisgenerator	→ sound 6,...
7			C ruis	B ruis	A ruis	C geluid	B geluid			→ sound 7,...
8				A effect					volume kanaal A	→ sound 8,...
9				B effect					volume kanaal B	→ sound 9,...
10				C effect					volume kanaal C	→ sound 10,...
11									tijdsduur effectontwikkeling	→ sound 11,...
12										→ sound 12,...
13									soort effect	→ sound 13,...

Elk vakje stelt de mogelijke plaatsing van een binair cijfer voor.

In register 0 en 1 is de toonhoogte voor het eerste van de in totaal drie geluidskanalen gekodeerd. Wanneer we de toonhoogte in Hz (trillingen per seconde) weten, kunnen we de waarde van regsister 0 en 1 als volgt berekenen:

$$\text{vulling register 0 en 1} = \frac{111760}{\text{aantal trillingen per seconde (Hz)}}$$

De uiteindelijke waarde voor register 0 en 1 dient over de twee registers te worden verdeeld. Hiertoe dienen we de waarde eerst in binaire (tweetallige) vorm te gieten. De acht rechtercijfers van het binaire getal gaan in register 0, de vier overige cijfers gaan in register 1. Wanneer we minder dan 12 cijfers hebben, vullen we het binaire getal gewoon links aan met nullen.

Een voorbeeld: stel dat we de PSG over het eerste kanaal een toon willen laten voortbrengen van 1225 trillingen per seconde (een vrij normale toon). De vulling voor registers 0 en 1 laat zich als volgt berekenen:

$$\text{vulling} = \frac{111760}{1225} = 91.2326\ldots$$ we ronden dit af naar 91.

91 decimaal is gelijk aan 1011011 binair (probeer maar via PRINT BIN$(91)).

We moeten dit binaire resultaat eerst naar 12 cijfers aanvullen: 000001011011

De eerste vier cijfers (0000) moeten in register 1 komen, de overige cijfers (01011011) komen in register 0.

Met het SOUND-kommando kunnen we een register direkt vullen. Hiertoe vermelden we achter het SOUND-kommando eerst het gewenste register, gevolgd door de waarde die in dat register moet komen.

Met de twee regels:

```
10 SOUND 1,&B0000          of 10 SOUND 1,0
20 SOUND 0,&B01011011      of 20 SOUND 0,91
```

vullen we de vereiste registers. In de eerste kolom zijn de waarden binair opgenomen; in de tweede kolom zijn ze omgerekend naar deci-

male waarden. Beide vormen zijn toegestaan.

Wanneer we het hierboven opgenomen programma laten RUNnen, gebeurt er niets. Helaas is het niet voldoende om alleen deze registers aan te spreken; er dienen nog meer zaken te worden geregeld. Maar hierover later.

Het zal duidelijk zijn dat de registers 2-3 en 4-5 dezelfde funktie hebben als register 0-1 maar dan voor de tweede en de derde stem (het tweede en derde geluidskanaal).

Met de PSG kunnen we ook ruis opwekken. Deze ruis, in kombinatie met andere effekten, stelt ons in staat om geweerschoten, helikopters, stoomlokomotieven etcetera na te bootsen. De PSG laat de ruis wat betreft de toonhoogte wat sturen. Een ruis met lage toonhoogte kan men bewerkstelligen door in register 6 een nulwaarde te stoppen. Een ruis met een hoge toonhoogte verkrijgt men wanneer register 6 met 11111 binair (31 decimaal) wordt gevuld. Voor een hoge ruis kunnen we bijvoorbeeld

`30 SOUND 6,&B11111` of `30 SOUND 6,31`

programmeren. Kijk nog eens op het schema van registers en merk op dat we steeds één vakje met één binair cijfer (0 of 1) vullen.

Register 7 bepaalt de werkelijke door de PSG te ondernemen aktie. Met dit register kunnen we de diverse geluids- en ruiskanalen aan- of uitzetten. Een ruis- of geluidskanaal zetten we aan door een binaire 0 en weer uit door en binaire 1. Wanneer we (zie het schema) alle zes vakjes van een 1 voorzien

`40 SOUND 7,&B111111` of `40 SOUND 7,63`

dan schakelen we de hele PSG uit en horen niets. Wanneer we bijvoorbeeld echter op het eerste kanaal een toon willen laten horen en op het tweede kanaal een ruis willen laten klinken, dan dienen die betreffende kanalen met een binaire 0 aan te worden gezet:

`40 SOUND 7,&B101110` of `40 SOUND 7,45`

Wanneer het bovenstaande wordt uitgevoerd, maakt u een kans dat er al wat te horen is. Zet u het met CONTROL-STOP eventueel maar weer uit.

Register 8 bepaalt het volume van het eerste kanaal. Register 9 en 10 doen hetzelfde voor het tweede en derde kanaal.

Het volume staat uit wanneer het betreffende register met 0000 binair wordt gevuld. Volop staat het volume bij een 1111 binair (15 decimaal). Het volume van kanaal 1 kunnen we dus volop zetten met:

```
50 SOUND 8,&B1111
```
of
```
50 SOUND 8,15
```

Alle tussenwaarden korresponderen natuurlijk met tussenliggende volumes.

Wanneer in register 8 binair een 10000 (16 decimaal) wordt geplaatst, dan wordt in dat register (zie schema) het effekt-plaatsje op 1 gezet. Op dat moment is het volume niet meer van belang; een speciaal geluidseffekt werd voor het eerste kanaal gezet. Op dezelfde wijze kan ook voor de kanalen 2 en 3 voor een effekt worden gekozen.

Het effekt zelf bepalen we in register 13. De effekten zijn genummerd van 0 tot en met 15 (binair van 0000 tot en met 1111). Sommige verschillende invullingen van register 13 komen in effekt met elkaar overeen. De mogelijke effekten zijn in het volgende schema opgenomen.

In dit schema zijn de gedragingen aangegeven van het toonvolume bij de betreffende effekten. Willen we een tokkelende (steeds weer wegstervende) toon hebben, dan programmeren we bijvoorbeeld ondermeer:

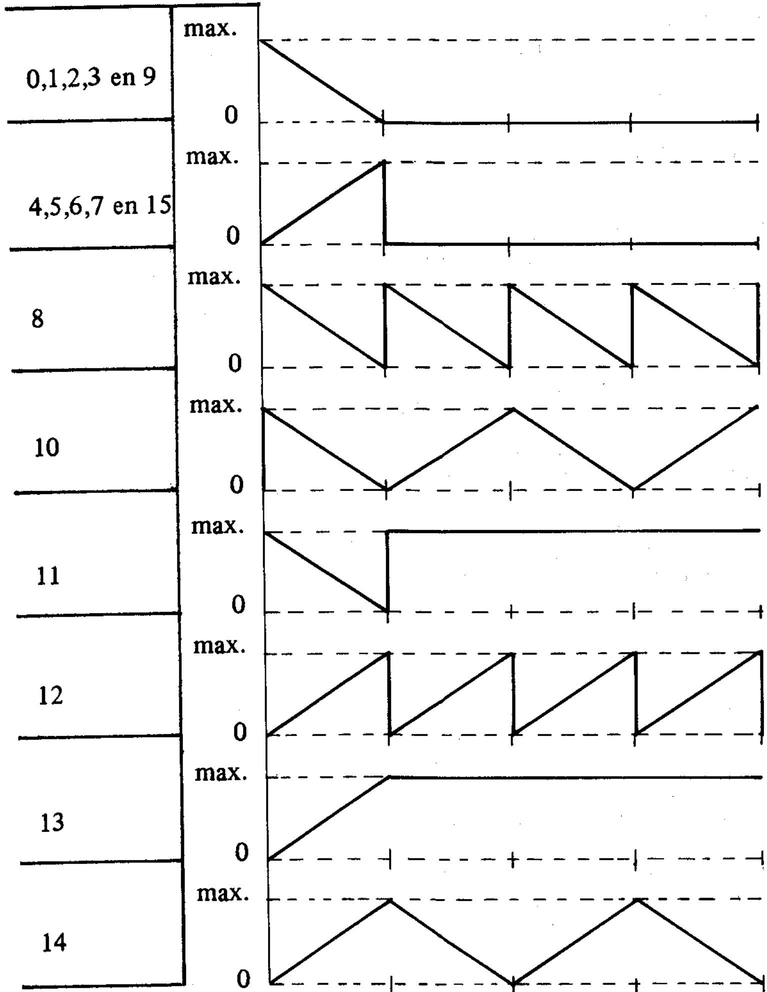
```
60 SOUND 13,&B1000
```
of
```
60 SOUND 13,8
```

In register 11 en 12 kunnen we tenslotte bepalen hoe lang het moet duren voordat het effekt éénmaal heeft plaatsgevonden. Dit doen we als volgt: we bepalen deze tijdsduur eerst in seconden. Daarna vermenigvuldigen we dit aantal seconden met het getal 6965. De zo verkregen uitkomst dient in register 11 en 12 te worden opgenomen.

Wanneer we het tokkelende effekt zo willen besturen dat er tien maal per seconde een toon wordt aangeslagen, dan programmeren we dat bijvoorbeeld als volgt:

Tijdsduur van het effekt = 1/10 seconde (we willen het effekt tien maal per seconde horen).

DE GELUIDSEFFECTEN



0.1 seconde maal 6965 = 696

695 decimaal is gelijk aan 1010111000 binair (probeer maar met BIN$).

Register 11 en 12 hebben samen 16 posities. We moeten de verkregen waarde dus eerst links met nullen aanvullen tot 16 posities:

0000001010111000

Dit binaire getal kunnen we in tweeën splitsen:

00000010 en 10111000

Het rechtergedeelte moet in register 11 en het linkergedeelte in register 12. Om de tijdsduur van 1/10 seconde te programmeren, kunnen we het volgende doen:

```
80 SOUND 11,&B10111000 of 80 SOUND 11,184
90 SOUND 12,&B00000010 of 90 SOUND 12,2
```

Door hele korte tijdsduren te koderen, kunnen we heel speciale effekten verkrijgen.

Wanneer we op deze manier alle registers aansturen, kunnen we geluid maken. Om te voorkomen dat we worden lastiggevallen met geluid dat nog niet helemaal goed is, moeten we de volgende volgorde aanhouden bij het programmeren van de PSG:

1. zet met behulp van register 7 alle kanalen uit (SOUND 7,63)
2. zet de gewenste toonhoogten (indien van toepassing) in register 0-6
3. stel de volumes of effekten in met behulp van register 8-10
4. zet de tijdsduur effektontwikkeling (register 12 en 13) op de juiste waarde (alleen indien van toepassing natuurlijk)
5. zet het gewenste effekt aan (indien van toepassing) in register 13
6. zet met behulp van register 7 de juiste kanalen open.

Een voorbeeld: programmering van een kanonschot:

eerst alle kanalen uitzetten	10 SOUND 7,&B111111
bepaal de toonhoogte van de ruis	20 SOUND 6,25
zet kanaal 1 op speciaal effekt	30 SOUND 8,&B10000
bepaal de tijdsduur van het effekt	40 SOUND 11,&B01101010

(2 seconden) 50 SOUND 12,&B00110110
bepaal het effekt (nr. 1) 60 SOUND 13,1
en zet ruiskanaal 1 open 70 SOUND 7,&B110111

Bovenstaand programma geeft het idee van het geluid van een kanonschot, wegstervend in 2 seconden. Door gebruik te maken van andere effekten en andere tijdsduren, kunnen andere geluiden worden gesimuleerd. Hieronder volgen twee voorbeelden: een helikopter en een stoomlokomotief.

```
10 SOUND 7, 63
20 SOUND 6, 15
30 SOUND 8, 16
40 SOUND 9, 16
50 SOUND 10, 12
60 SOUND 11, 92
70 SOUND 12, 1
80 SOUND 13, 10
90 SOUND 7, 7
```

```
10 SOUND 7, 63
20 SOUND 6, 31
30 SOUND 8, 16
40 SOUND 9, 16
50 SOUND 11, 184
60 SOUND 12, 2
70 SOUND 13, 14
80 SOUND 7, 39
```

Omdat het programmeren van de PSG waarschijnlijk altijd wel een vervelende en moeilijke klus zal blijven, is hieronder een programma opgenomen dat u heel veel werk kan besparen.
Het programma heet Sound Generator Editor en vraagt u om diverse gegevens. Geef steeds JA of NEE of een getalswaarde in wanneer het programma u iets vraagt.

Naar aanleiding van de beantwoording van de aan u gestelde vragen, stelt dit programma de registerinstelling samen en laat u het bijbehorende effekt horen.

Wanneer u tevreden bent met het effekt, kunt u JA antwoorden op de vraag of u dit effekt geprogrammeerd wilt hebben. In dat geval wist het programma zichzelf nadat het de registerinstellingen op beeld heeft geplaatst. U hoeft dan eigenlijk alleen nog maar de regelnummers in te geven en het hele gebeuren is perfekt geprogrammeerd.

Vergeet niet om het programma eerst te SAVEn alvorens het uit te proberen. Mocht er iets fout gaan, dan bent u het in ieder geval niet kwijt.

```
10 REM ***************************
20 REM * SOUND REGISTER EDITOR *
30 REM ***************************
40 REM
50 CLS:SCREEN 0:WIDTH 40:PRINT "SOUND-GE
NERATOR EDITOR":PRINT
60 DEF FNA(A$)=(A$="" OR LEFT$(A$,1)="N"
 OR LEFT$(A$,1)="n"):INPUT "WILT U GELUI
DSKANALEN ACTIVEREN ";A$
70 IF FNA(A$) THEN 120
80 PRINT:PRINT"GEEF DE TOONHOOGTEN IN Hz
 (AANTAL TRIL- LINGEN PER SECONDE) IN VO
OR DE DRIE KA- NALEN (0=GEEN, ANDERS TUS
SEN 28-111760)":PRINT
90 INPUT "TOONHOOGTE KANAAL 1 ";T1:IF T1
>0 AND T1<28 OR T1>111760! THEN 90
100 INPUT "TOONHOOGTE KANAAL 2 ";T2:IF T
2>0 AND T2<28 OR T2>111760! THEN 100
110 INPUT "TOONHOOGTE KANAAL 3 ";T3:IF T
3>0 AND T3<28 OR T3>111760! THEN 110
120 PRINT:A$="":INPUT "WILT U EEN RUISEF
FECT ";A$
130 IF FNA(A$) THEN TR=0:GOTO 190
140 PRINT:PRINT "GEEF DE TOONHOOGTE VAN
DE GEWENSTE RUIS IN (0-31, 0=LAAG, 31=HO
OG)":PRINT
150 INPUT "TOONHOOGTE VAN DE RUIS ";TR:I
F TR<0 OR TR >31 THEN 150 ELSE PRINT
```

```
160 A$="":INPUT "WILT U RUIS OVER KANAAL
 1 ";A$:IF FNA(A$)=0 THEN R1=1
170 A$="":INPUT "WILT U RUIS OVER KANAAL
 2 ";A$:IF FNA(A$)=0 THEN R2=1
180 A$="":INPUT "WILT U RUIS OVER KANAAL
 3 ";A$:IF FNA(A$)=0 THEN R3=1
190 IF T1+T2+T3+R1+R2+R3=0 THEN RUN
200 PRINT:PRINT "DE VOLGENDE EFFEKTEN KU
NNEN WORDEN GE-  KOZEN:"
210 PRINT:PRINT "0:-----  KONSTANTE TOON
"
220 PRINT "1:\_____ WEGSTERVEND"
230 PRINT "2:/!___  OPKOMEND EN PLOTSELI
NG WEG"
240 PRINT "3:\!\!\  WEGSTERVEND, STEEDS
WEER TERUG"
250 PRINT "4:\/\/\  STEEDS WEGSTERVEND E
N OPKOMEND"
260 PRINT "5:\!---  WEGSTERVEND,PLOTS AA
NHOUDEND"
270 PRINT "6:/!/!/  OPKOMEND,PLOTS WEG,
HERHALEND"
280 PRINT "7:/----  OPKOMEND EN AANHOUDE
ND"
290 PRINT "8:/\/\/  STEEDS OPKOMEND EN W
EGSTERVEND":PRINT
300 INPUT "EFFEKT ";EF:IF EF<0 OR EF>8 T
HEN 300
310 IF EF=0 THEN GOTO 350 ELSE PRINT
320 IF T1+R1 THEN A$="":INPUT "WILT U DI
T EFFEKT OP KANAAL 1 ";A$:IF FNA(A$)=0 T
HEN F1=1
330 IF T2+R2 THEN A$="":INPUT "WILT U DI
T EFFEKT OP KANAAL 2 ";A$:IF FNA(A$)=0 T
HEN F2=1
340 IF T3+R3 THEN A$="":INPUT "WILT U DI
```

```
T EFFEKT OP KANAAL 3 ";A$:IF FNA(A$)=0 T
HEN F3=1
350 IF (T1+R1=0 OR F1=1) AND (T2+R2=0 OR
 F2=1) AND (T3+R3=0 OR F3=1) THEN GOTO 3
90 ELSE PRINT:PRINT "U KUNT VOOR ELK KAN
AAL HET VOLUME INGE- VEN. 0=GEEN VOLUME,
 15=HOOGSTE VOLUME":PRINT
360 IF T1+R1>0 AND F1=0 THEN INPUT "VOLU
ME KANAAL 1 ";V1:IF V1<O OR V1>15 THEN 3
60
370 IF T2+R2>0 AND F2=0 THEN INPUT "VOLU
ME KANAAL 2 ";V2:IF V2<O OR V2>15 THEN 3
70
380 IF T3+R3>0 AND F3=0 THEN INPUT "VOLU
ME KANAAL 3 ";V3:IF V3<O OR V3>15 THEN 3
80
390 IF F1+F2+F3=0 THEN 420 ELSE PRINT
400 PRINT"GEEF NU DE TIJDSDUUR IN SECOND
EN IN      WAARBINNEN HET INGEGEVEN EFFEK
T EENMAAL DIENT TE ZIJN OPGETREDEN. U MA
G CIJFERS NA DE KOMMA INGEVEN (MAX 9.4 S
ECONDEN)":PRINT
410 INPUT "EFFEKT-DUUR ";ED:IF ED<O OR E
D>9.4 THEN 410
420 PRINT
430 G0=0:G1=0:IF T1 THEN G=111760!/T1:G0
=G MOD 256:G1=G\256
440 G2=0:G3=0:IF T2 THEN G=111760!/T2:G2
=G MOD 256:G3=G\256
450 G4=0:G5=0:IF T3 THEN G=111760!/T3:G4
=G MOD 256:G5=G\256
460 G6=0:IF TR>0 THEN G6=TR
470 G7=63-SGN(T1)-2*SGN(T2)-4*SGN(T3)-8*
R1-16*R2-32*R3
480 GD=(EF+6+(EF<4)+3*(EF<3)+4*(EF<2))*(
EF>1)*-1
```

```
490 G8=V1+16*F1:G9=V2+16*F2:GA=V3+16*F3
500 G=INT(ED*6965):GB=G-256*INT(G/256):G
C=INT(G/256)
510 SOUND 7,63:SOUND 0,G0:SOUND 1,G1:SOU
ND 2,G2:SOUND 3,G3:SOUND 4,G4:SOUND 5,G5
:SOUND 6,G6:SOUND 8,G8:SOUND 9,G9:SOUND
10,GA:SOUND 11,GB:SOUND 12,GC:SOUND 13,G
D:SOUND 7,G7
520 A$="":INPUT "U HOORT NU HET EFFEKT.
NOGMAALS ";A$
530 IF FNA(A$)=0 THEN 510
540 CLS:BEEP:PRINT:PRINT "ZO PROGRAMMEER
T U DIT EFFEKT:":PRINT:PRINT
550 PRINT "XXXX SOUND 7, 63"
560 IF T1 THEN PRINT "XXXX SOUND 0,";G0:
PRINT "XXXX SOUND 1,";G1
570 IF T2 THEN PRINT "XXXX SOUND 2,";G2:
PRINT "XXXX SOUND 3,";G3
580 IF T3 THEN PRINT "XXXX SOUND 4,";G4:
PRINT "XXXX SOUND 5,";G5
590 IF R1+R2+R3 THEN PRINT "XXXX SOUND 6
,";G6
600 IF T1+R1 THEN PRINT "XXXX SOUND 8,";
G8
610 IF T2+R2 THEN PRINT "XXXX SOUND 9,";
G9
620 IF T3+R3 THEN PRINT "XXXX SOUND 10,"
;GA
630 IF F1+F2+F3 THEN PRINT "XXXX SOUND 1
1,";GB:PRINT"XXXX SOUND 12,";GC:PRINT"XX
XX SOUND 13,";GD
640 PRINT"XXXX SOUND 7,";G7
650 PRINT:A$="":INPUT "WILT U DIT EFFEKT
 PROGRAMMEREN ";A$
660 IF FNA(A$) THEN RUN ELSE LOCATE 0,CS
RLIN-1:PRINT SPACE$(40):LOCATE 0,0:PRINT
```

```
"VERVANG XXXX DOOR DE JUISTE REGELNUMME
RSHET GEMAKKELIJKST BEEINDIGT U EEN EFFE
KTMET HET 'BEEP'-KOMMANDO":NEW
```

3 De video display processor

Net als de PSG is de Video Display Processor (kortweg VDP) een klein computertje binnen een computer. De VDP regelt al het zichbare van een MSX-computer (schermindelingen, sprites e.d.).

Met de SCREEN-kommando's, het COLOR-kommando en nog een handvol andere kommando's kunnen we de VDP via MSX-basic besturen. Ook met de VDP kunnen we echter een beetje 'buiten basic om' programmeren en dan kunnen we ineens veel grappige dingen doen.

De VDP heeft een apart stuk geheugen binnen de MSX-computer van 16 kilobytes groot. Dit VRAM (Video Random Access Memory) is nodig om de beeldscherminformatie in te kunnen opslaan.

De indeling van het VRAM kan binnen MSX-basic met behulp van de BASE-systeemvariabele worden opgevraagd en bestuurd.

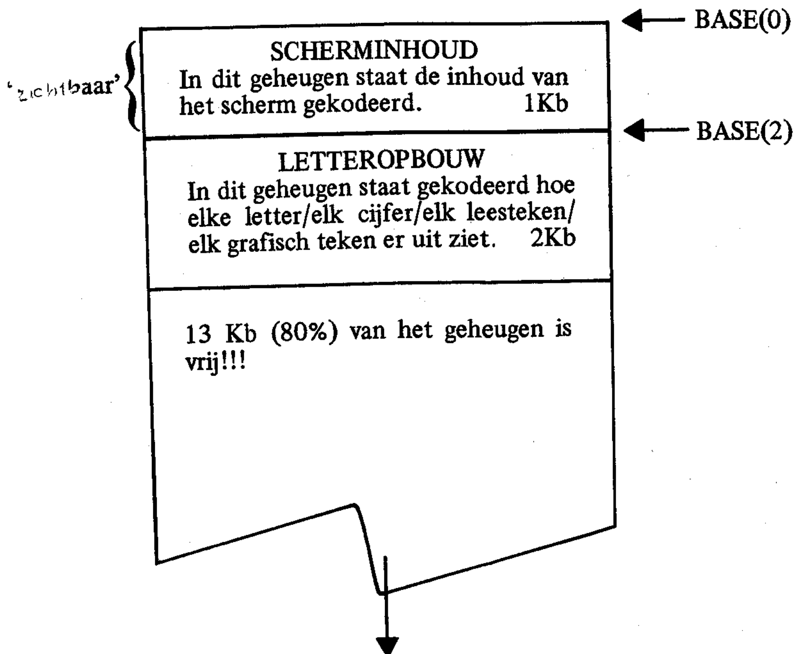
Voordat we wat gaan goochelen binnen het VRAM, behandelen we hier eerst de indeling van het VRAM bij de SCREEN 1 en de SCREEN 0 instelling.

In het volgende schema kunnen we bij de SCREEN 0 instelling binnen het VRAM gebieden onderscheiden.

Onmiddellijk valt op dat 80% van het VRAM bij een SCREEN 0-instelling ongebruikt blijft!

De systeemvariabelen BASE(0) en BASE(2) verwijzen naar de verschillende geheugendelen binnen het VRAM.

BASE(0) verwijst naar het ZICHTBARE VRAM-geheugengedeelte. Alle informatie die in dit geheugen is gekodeerd, is ook op het beeld-



scherm te zien.

Met een VPOKE-instruktie kan het VRAM worden veranderd terwijl met de VPEEK-instruktie het VRAM kan worden uitgelezen. Om te laten zien dat BASE(0) werkelijk verwijst naar het zichtbare geheugengedeelte, kunnen we de volgende test doen:

tik in:
```
SCREEN 0
Ok
VPOKE BASE(0),ASC("Q")
Ok
```

Bij het intikken van de tweede instruktie verschijnt boven in het beeld plotseling de letter Q. De kode van deze letter, weergegeven door ASC("Q"), werd iets voorbij het geheugenadres BASE(0) in het VRAM geplaatst.

Het geheugengedeelte waarnaar BASE(2) verwijst, is veel interessanter! In dat geheugengedeelte staat namelijk de karakteropbouw geko-

deerd. Het te gebruiken LETTERTYPE is hier als het ware opgenomen.

Het volgende progamma leest het VRAM vanaf het adres BASE(2) uit en projekteert de gevonden waarden BINAIR op het beeldscherm. Wanneer u het programma laat werken, ziet u, gekodeerd in nullen en enen, alle MSX-karakters één voor één sterk vergroot op het beeldscherm komen:

```
10 REM ******************************
20 REM * KARAKTERS BINAIR OP BEELD *
30 REM * (STOP=TIJDELIJK ONDERBR.) *
40 REM ******************************
50 REM
60 SCREEN 0:FOR I=BASE(2) TO BASE(2)+204
7:IF I MOD 8=0 THEN PRINT
70 PRINT RIGHT$(BIN$(VPEEK(I)+256),8)
80 NEXT I
```

Zoals het bovenstaande programma liet zien, is bijvoorbeeld de letter A als volgt in het VRAM gekodeerd:

```
0 0 1 0 0 0 0 0
0 1 0 1 0 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
1 1 1 1 1 0 0 0
1 0 0 0 1 0 0 0
1 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
```

8 x 8

Door het VRAM op de juiste wijze te ontleden, kan het lettertype zichtbaar worden gemaakt van de MSX-karakters.

Wat leuker is, is dat dit lettertype ook kan worden veranderd. Probeer het volgende programma maar eens:

```
10 REM ******************
20 REM * ANDERE LETTER A *
30 REM ******************
40 REM
```

```
50 DATA 248,136,136,248,136,136,136,136
60 FOR I=BASE(2)+520 TO BASE(2)+527:READ
 B:VPOKE I,B:NEXT I
```

In het bovenstaande voorbeeld werd de hoofdletter A in een ander jasje gestoken. Elke hoofdletter A die we nu intikken, ziet er voortaan wat anders uit.

De oude lettertypes kunnen we overigens weer terug krijgen door de computer een keer uit en weer aan te zetten of door een SCREEN-kommando (bijvoorbeeld SCREEN 0) in te tikken.

De VRAM-geheugenindeling bij een SCREEN 1-instelling lijkt veel op de indeling van het VRAM-geheugen bij SCREEN 0. Echter, de rol BASE(2) wordt nu door BASE(7) overgenomen en de rol van BASE(0) wordt door BASE(5) overgenomen. BASE(6) verwijst naar een extra tabel waarin de verschillende kleuren zijn gekodeerd die in een SCREEN 1-instelling kunnen worden toegepast.

Nu we een beetje weten hoe de VRAM-indeling er uit ziet, volgen hier eerst twee tips voor de ondernemende programmeur:

Vaak is de amateur die iets verder is gevorderd, op een gegeven moment op zoek naar extra geheugen om zijn gegevens op te slaan. Het normale computergeheugen is niet meer toereikend en er moet een alternatief worden gevonden.
We zagen reeds dat in een SCREEN 0-indeling ongeveer 80% van het VRAM-geheugen niet wordt gebruikt; 13 kilobytes geheugen die uitstekend van pas kunnen komen.

Door een juist gebruik te maken van de BASE(0) en BASE(2) systeemvariabelen, kan MSX worden gedwongen, een bepaald stuk geheugen in het VRAM te bezetten. Het overige geheugen is dan helemaal vrij; de programmeur kan op die plaats zijn gegevens VPOKEn en ze later weer met VPEEK terughalen.

Het volgende stukje programmatuur maakt dat in het VRAM de geheugenplaatsen 0 tot en met 13311 vrij kunnen worden gebruikt:

```
10 BASE(2)=14336:BASE(0)=13312:SCREEN0
```

Het flinke stuk geheugen dat in de SCREEN 0-instelling ongebruikt blijft, kan ook op een andere manier worden gebruikt. Onder SCREEN

0 kunnen we maximaal 14 schermen onafhankelijk van elkaar opslaan in VRAM. Door steeds met BASE(2) te manipuleren, kunnen we een ander scherm zichtbaar maken. Behalve het veranderen van BASE(0) is er ook een geheimzinnige POKE nodig om het MSX-basic duidelijk te maken dat een ander scherm nu aktief is.

Het volgende programma kan als subroutine in een groter geheel worden opgenomen. De eerste maal dat de routine (GOSUB 10000) wordt aangeroepen, worden alle 14 schermen schoongemaakt. Elke volgende keer dat de subroutine wordt aangesproken, moet de variabele SC de waarde 2 tot en met 16 hebben. Het corresponderende scherm wordt dan geaktiveerd. Ook eventuele kursorposities worden bewaard en hersteld.

Zo sterk is MSX-basic als je het weet te gebruiken. Gewoon in basic, met maar een paar POKE-statements, maken we een supersnel schermwissel-programma.

```
10 REM ***************************
20 REM * 14 SCHERMEN SCREEN 0    *
30 REM * ZONDER GEHEUGENVERLIES *
40 REM * 1E KEER GOSUB 10000 IS *
50 REM * SCHOONMAKEN ALLE SCHER-*
60 REM * MEN. VOLGENDE KEREN IS *
70 REM * IS OVERSCHAKELEN NAAR   *
80 REM * SCHERM NUMMER SC (2/15)*
90 REM * OOK DE BIJBEHORENDE     *
100 REM* CURSORPOSITIE WORDT     *
110 REM* WEER INGESTELD.         *
120 REM***************************
130 REM
10000 IF OC=0 THEN BASE(2)=0:FOR OC=15 T
O 2 STEP -1:BASE(0)=OC*1024:SCREEN 0:CLS
:VPOKE OC*1024+1022,0:VPOKE OC*1024+1023
,0:NEXT OC:OC=2:RETURN
10010 VPOKE OC*1024+1022,POS(0):VPOKE OC
*1024+1023,CSRLIN:BASE(0)=SC*1024:POKE 6
3779!,4*SC:OC=SC:LOCATE VPEEK(OC*1024+10
22),VPEEK(OC*1024+1023):RETURN
```

Zoals we al lieten zien, kan het lettertype van een MSX-computer gemakkelijk worden veranderd. Echter, deze verandering verdwijnt wanneer de computer uit wordt gezet of wanneer een SCREEN-opdracht wordt gebruikt.

MSX voorziet in 255 tekens. Maar een klein gedeelte van deze tekens wordt gevormd door hoofdletters, kleine letters, cijfers en leestekens.

In veel gevallen is het gewenst dat we onze eigen lettertekens of grafische tekens kunnen ontwerpen. We hebben dan wel een programma nodig dat ons in staat stelt om op een gemakkelijke wijze de bestaande tekens te veranderen.

Dat programma is hieronder opgenomen en heet Character Editor. Nadat het programma is opgestart, verschijnen onder in het beeld alle mogelijke MSX-karakters en kunnen we het te veranderen karakter ingeven. Nadat we het karakter hebben ingegeven, wordt het vergroot afgebeeld en kunnen we met de pijltoetsen naar elk puntje binnen het gekozen karakter gaan. Door de spatietoets in te geven, kunnen we een puntje inkleuren of juist leeg maken. Op het beeldscherm wordt het kleine karakter onmiddellijk ook aangepast zodat op elk moment een normaal beeld kan worden gevormd van het karakter dat we aan het ontwerpen zijn.

Door uiteindelijk een RETURN in te geven, gaan we weer naar de ingave van het te veranderen karakter. Op deze wijze kunnen we op eenvoudige wijze ons eigen lettertype samenstellen.

Door bij de ingave van het te veranderen karakter geen karakter in te geven (alleen RETURN), gaat het programma naar de ingave L/S. Geef een L in wanneer een set karakters moet worden ingelezen of een S in wanneer een set karakters op floppy of band moet worden geschreven.

Na deze keuze dient nog een FILENAAM te worden ingegeven. Onder deze naam wordt de set op band/floppy geschreven of juist opgehaald.

Wanneer we het programma willen beëindigen, geven we een normale CONTROL-STOP in. Het programma geeft eerst nog de instruktie hoe de door u samengestelde karakterset in een willekeurig programma kan worden opgenomen en is daarna beëindigd.

```
10 REM *************************
20 REM *    CHARACTER EDITOR    *
30 REM *************************
40 REM
50 ON STOP GOSUB 210:STOP ON
60 DEF FNA(X,S)=(S>1 AND S<5)*(X<7)-(S>5
)*(X>0):DEF FNB(Y,S)=(S>3 AND S<7)*(Y<7)
-(S=1 OR S=2 OR S=8)*(Y>0)
70 KEY OFF:BASE(7)=0:BASE(5)=2048:BASE(6
)=3072:SCREEN 1:CLS:WIDTH 32
80 FOR I=2024 TO 2039:VPOKE I,255:NEXT I
:FOR I=2025 TO 2030:VPOKE I,129:NEXT I
90 PRINT "KARAKTER :":PRINT "SAVE/LOAD:"
:PRINT "FILENAAM :"
100 FOR I=2560 TO 2812:VPOKE I,I-2560:NE
XT I
110 LOCATE 10,0,0:PRINT SPACE$(20):LOCAT
E 10,0:LINE INPUT K$:IF K$="" THEN 170 E
LSE LOCATE 23,14:PRINT K$;K$:IF LEN(K$)=
2 AND ASC(K$)=1 THEN K$=CHR$(ASC(MID$(K$
,2))-64)
120 B=ASC(K$)*8:FOR Y=0 TO 7:K=VPEEK(B+Y
):FOR X=0 TO 7:LOCATE X+20,Y+4:IF K AND
2^(7-X) THEN PRINT CHR$(254) ELSE PRINT
CHR$(253)
130 NEXT X:NEXT Y:X=0:Y=0
140 IF INKEY$=CHR$(13) THEN 110 ELSE LOC
ATE X+20,Y+4,M:M=-(M=0):IF STRIG(0)=0 TH
EN S=STICK(0):X=X+FNA(X,S):Y=Y+FNB(Y,S):
IF S THEN BEEP:FOR I=1 TO 50:NEXT I::GOT
O 140 ELSE 140
150 K=VPEEK(B+Y):IF K AND 2^(7-X) THEN K
=K XOR 2^(7-X):PRINT CHR$(253) ELSE K=K+
2^(7-X):PRINT CHR$(254)
160 VPOKE B+Y,K:BEEP:FOR I=1 TO 50:NEXT
I:GOTO 140
```

```
170 LOCATE 10,1,0:PRINT SPACE$(20):LOCAT
E 10,1:LINE INPUT K$:IF K$>"" AND K$<>"L
" AND K$<>"l" AND K$<>"S" AND K$<>"s" TH
EN 170 ELSE IF K$="" THEN 110
180 LOCATE 10,2:PRINT SPACE$(20):LOCATE
10,2:LINE INPUT F$:IF LEN(F$)>6 THEN 180
 ELSE IF K$="" THEN 170
190 IF K$="S" OR K$="s" THEN OPEN F$ FOR
 OUTPUT AS 1:FOR I=0 TO 2047:PRINT #1,RI
GHT$("00"+HEX$(VPEEK(I)),2);:NEXT I:CLOS
E ELSE OPEN F$ FOR INPUT AS 1:FOR I=0 TO
 2047:VPOKE I,VAL("&H"+INPUT$(2,1)):NEXT
 I:CLOSE
200 GOTO 110
210 SCREEN 0:PRINT:PRINT "LADEN ONDER SC
REEN 0:":PRINT:PRINT "XXXX BLOAD ...,S,B
ASE(2)":PRINT:PRINT:PRINT"LADEN ONDER SC
REEN 1:":PRINT:PRINT "XXXX BLOAD ...,S,B
ASE(7)":PRINT:PRINT:PRINT"... = NAAM WAA
RONDER IS GESAVED":PRINT:STOP
```

Hoe u de verschillende karakters via uw toetsenbord tevoorschijn kunt toveren, staat in de volgende tabellen vermeld:

TABEL A

1 2 3 4 5 6 7 8 9 0 - =
q w e r t y u i o p [ ]
a s d f g h j k l ; ' `
z x c v b n m , . / \

Karakters die normaal ontstaan bij het intoetsen.

TABEL B

! @ # $ % ^ & * ( ) _ +
Q W E R T Y U I O P { }
A S D F G H J K L : " ~
Z X C V B N M < > ? |

Karakters die ontstaan indien de toetsen tegelijk met de SHIFT-toets worden ingedrukt.

TABEL C

Karakters die ontstaan indien de toetsen tegelijk met de GRAFH-toets worden ingedrukt.

TABEL D

Karakters die ontstaan indien de toetsen tegelijk met de SHIFT- en GRAFH-toets worden ingedrukt.

TABEL E

Karakters die ontstaan indien de toetsen tegelijk met de CODE-toets worden ingedrukt.

TABEL F

Karakters die onstaan indien de toetsen tegelijk met de SHIFT- en CODE-toets worden ingedrukt.

4 Ontbinden in faktoren en breuken

Op de lagere en middelbare school hebben we allemaal geleerd wat ontbinden in faktoren is. Onbewust maakten we van deze techniek vaak gebruik bij het vereenvoudigen van breuken.

Grappig is het om te zien dat de computer-amateur, hoe geslepen hij of zij ook vaak is, vaak in de grootste moeite komt wanneer er bijvoorbeeld priemgetallen met behulp van de computer moeten worden bepaald.

Nog moeilijker wordt het wanneer een getal met behulp van de computer in faktoren moet worden ontbonden. Het werken met en het vereenvoudigen van breuken vormt voor de amateur vaak het grootste probleem.

Wanneer je voor je zoontje of dochtertje een breuken-programma moet schrijven en je slaagt daar als ouder en fervent computeramateur niet in, dan sla je natuurlijk wél een figuur.

Hieronder worden de onderwerpen: **ontbinden in faktoren en rekenen met breuken** uitgebreid behandeld. Niet alleen de programma's worden gegeven maar ook de manier waarop zij werken, wordt behandeld.

Met onderstaande stof bent u als computer-ouder (of misschien als onderwijzer of onderwijzeres?) weer even gewapend tegen de vragen van uw computer-jeugd...

4.1 Ontbinden in faktoren

Wanneer we een probleem aan de computer ter oplossing willen aanbieden, zullen we eerst zelf dit probleem moeten kunnen doorgronden. De akties die de computer dient te ondernemen, dient u eerst voor u zelf te kunnen formuleren. In geval van het ontbinden van een getal in

faktoren zoū de aktie die moet worden ontplooid, als volgt kunnen worden geformuleerd:

Hoe ontbind ik een getal in faktoren?

Algemeen:

Elk getal, behalve een priemgetal, kan worden geschreven als een vermenigvuldiging van twee of meer andere getallen. Ook deze getallen (behalve wanneer het uiteindelijk ondeelbare (priem-)getallen zijn) kunnen weer worden geschreven als een vermenigvuldiging van twee of meer andere getallen, enzovoorts.
Uiteindelijk kan elk getal worden uitgeschreven als een vermenigvuldiging van priemgetallen. Het bepalen van deze vermenigvuldiging heet 'ontbinden in faktoren.'

Om een getal te ontbinden in faktoren, moeten we steeds een volgend priemgetal bepalen en het getal daardoor trachten te delen waarna we eventueel met de uitkomst van deze deling verder gaan.

Voorbeeld: ontbind 660 in faktoren:

Priemgetal 1 doet niet mee. Het eerste priemgetal dat aan bod komt is dus 2.

660 gedeeld door 2 = 330 — *660 is gelijk aan 2 x 330*
330 gedeeld door 2 = 165 — *660 is gelijk aan 2 x 2 x 165*

165 is niet meer deelbaar door 2. Het volgende priemgetal is 3:

165 gedeeld door 3 = 55 — *660 is gelijk aan 2 x 2 x 3 x 55*

55 is niet meer deelbaar door 3. Het volgende priemgetal is 5:

55 gedeeld door 5 = 11 — *660 is gelijk aan 2 x 2 x 3 x 5 x 11*

11 is niet meer deelbaar door 5. Ook door het volgende priemgetal, 7, is 11 niet meer deelbaar. Het volgende priemgetal is 11; 11 is dus zélf een priemgetal. Het getal 660 is hiermee dus in faktoren ontbonden:

660 = 2 x 2 x 3 x 5 x 11 *ofwel* $660 = 2^2 \times 3 \times 5 \times 11$

Meestal wordt bij het ontbinden in faktoren een schema getekend, ongeveer als volgt:

660	2
330	2
165	3
55	5
11	11
1	

Steeds wordt rechts van de streep het getal vermeld waardoor wordt gedeeld en links, één stap lager, het resultaat ná deling.

Uiteindelijk, wanneer het schema op 1 uitkomt, is rechts de ontbinding in faktoren af te lezen.

Aan de hand van de volgende veel uitgebreidere beschrijving kunnen we gemakkelijk een MSX-basic-programma maken dat getallen in faktoren ontbindt.

Een aktieschema zoals het hieronder staat, dat punt voor punt precies aangeeft welke akties moeten worden ondernomen teneinde een bepaald probleem op te lossen, noemt men een ALGORITHME.

Het ontwerpen van een ALGORITHME gaat altijd vooraf aan het ontwerpen van een programma.

Het algorithme voor het ontbinden van een getal in faktoren kan er als volgt uitzien:

Hoe ontbind ik een getal in faktoren?

Algorithme:

1 *we gaan uit van GETAL*
2 *stel DELER op de waarde 2*
3 *wanneer GETAL kleiner is dan DELER of gelijk is aan DELER dan is het getal ontbonden in faktoren en is de berekening ten einde*
4 *wanneer GETAL deelbaar is door DELER, deel GETAL dan ook daadwerkelijk door DELER en noteer deze DELER als faktor. Ga terug naar punt 3*
5 *wanneer DELER is gelijk aan 2, stel DELER dan op 3. Verhoog*

DELER anders met 2
6 ga terug naar punt 3

Aan de hand van dit algorithme kunnen we dan uiteindelijk een MSX-basic-programma ontwerpen. Het volgende programma ontbindt een ingegeven getal in faktoren.

```
10 REM **************************
20 REM * ONTBINDEN IN FAKTOREN *
30 REM **************************
40 REM
50 DEF FNU$(X)=MID$(STR$(X),2)
60 CLS:INPUT "IN FAKTOREN ONTBINDEN ";F:
IF F=0 THEN STOP ELSE IF F<2 OR F>INT(F)
 THEN BEEP:RUN
70 CLS:PRINT:PRINT FNU$(F);" =";:P=2
80 T=0:IF F=1 THEN 120
90 IF F/P=INT(F/P) THEN T=T+1:F=F/P:GOTO
 90
100 IF T THEN PRINT STR$(P);CHR$(30);FNU
$(T);STRING$(-(T=1),8);STRING$(-(T=1),"
");CHR$(10);"x";
110 P=P+2+(P=2):GOTO 80
120 PRINT CHR$(8);" ":PRINT:PRINT "GEEF
EEN TOETS IN"
130 IF INKEY$="" THEN 130 ELSE RUN
```

4.2 Rekenen met breuken

Wanneer we in MSX-basic met breuken willen gaan werken, komen we al snel op het probleem:

Hoe geef je in MSX-basic breuken in?

Met de computer kun je prima rekenen zolang getallen met decimalen worden ingegeven en ook weer met decimalen mogen worden verstrekt. Moet er echter niet met decimalen maar met breuken worden gewerkt, dan krijgt elke computer het moeilijk. Wanneer we met breuken willen werken, zullen we dat zélf helemaal in basic moeten regelen.

Voordat we met breuken kunnen gaan werken, moeten we in staat zijn om breuken aan de computer te kunnen opgeven.

De volgende subroutine is in alle verdere breuk-voorbeelden terug te vinden en regelt de komplete breuk-ingave voor ons. Het enige dat we in het hoofdprogramma behoeven te doen, is een GOSUB opnemen naar deze routine nadat we de variabele I$ hebben gevuld met de ingavepositie op beeldscherm en de voor de ingave af te drukken tekst. De GOSUB verzorgt ingave en kontrole en vult uiteindelijk voor ons de variabelen B1, B2 en B3 met achtereenvolgens het aantal ingegeven GEHELEN, de ingegeven TELLER en de ingegeven NOEMER.

De onderstaande routine staat ons toe, de breuken op de volgende manieren in te geven:

Ingavevoorbeeld	(gehelen) B1	(teller) B2	(noemer) B3	Opmerking
1 2/3	1	2	3	normale ingave
12	12	0	0	ook een geheel getal geldt als een 'breuk'
22/77	0	22	77	gehelen behoeven niet te worden opgegeven

Breuken-ingaveroutine:

```
10 REM ***************************
20 REM * DEZE SUBROUTINE VER-    *
30 REM * ZORGT DE INGAVE VAN     *
40 REM * EEN BREUK. DE BREUK     *
50 REM * KAN OP ELKE GEWENSTE    *
60 REM * WIJZE WORDEN INGEGE-    *
70 REM * VEN ZOLANG DE BREUK-    *
80 REM * STREEP MAAR WORDT GE-   *
90 REM * VORMD DOOR EEN "/"      *
100 REM*                         *
110 REM* I$="XXYYA---A"          *
120 REM* XX/YY=DE GEWENSTE PO-   *
```

```
130 REM* SITIE OP BEELDSCHERM   *
140 REM* A--A=DE VOOR DE INGA-  *
150 REM* VE AF TE DRUKKEN       *
160 REM* TEKST                  *
170 REM*                        *
180 REM* IN B1 KOMT HET GEHELE  *
190 REM* DEEL VAN DE BREUKNO-   *
200 REM* TATIE, IN B2 KOMT DE   *
210 REM* WAARDE VAN DE TELLER   *
220 REM* EN IN B3 DE WAARDE     *
230 REM* VAN DE NOEMER          *
240 REM*************************
250 REM
260 LOCATE VAL(LEFT$(I$,2)),VAL(MID$(I$,
3,2)):PRINT MID$(I$,5);
270 B1=0:B2=0:B3=0:E=0:LINE INPUT B$:B0$
=B$:IF B$="" THEN E=1:RETURN ELSE FOR II
=1 TO LEN(B$):BB$=MID$(B$,II,1):IF (BB$<
"0" OR BB$>"9") AND BB$<>"/" AND BB$<>"
" THEN 260 ELSE NEXT II
280 P=INSTR(B$," "):IF P>1 THEN B1=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1) ELSE IF P
THEN 260 ELSE IF INSTR (B$,"/")=0 THEN B
1=VAL(B$):RETURN
290 P=INSTR(B$,"/"):IF P>1 THEN B2=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1):IF LEN(B$)
 THEN B3=VAL(B$):GOTO 300 ELSE 260 ELSE
260
300 IF B2>0 AND B3=0 THEN 260 ELSE RETUR
N
```

4.3 Breuken vereenvoudigen

De subroutine BREUK-INGAVE die we eerder behandelde, komt in alle volgende programma's voor; het hoofdprogramma bevat alleen het rekenwerk met breuken en blijft steeds tot een paar regels beperkt.

Wanneer we voor ons zelf moeten formuleren hoe een breuk moet worden vereenvoudigd, kunnen we dat bijvoorbeeld als volgt doen:

Hoe vereenvoudig ik een breuk?

Algemeen:

Allereerst moeten we er voor zorgen dat de gehelen uit de breuk worden gehaald. Bijvoorbeeld: 22/7 is gelijk aan 3 1/7 (drie gehelen plus 1/7).
Daarna moeten we van het breukgedeelte de teller en noemer net zolang door steeds hetzelfde getal delen totdat deze zo klein mogelijk geworden zijn.
Bijvoorbeeld: 4/6 is gelijk aan 2/3 (teller en noemer door 2 gedeeld).

Voor de computer ontwerpen we een wat exaktere omschrijving; het algorithme:

Hoe vereenvoudig ik een breuk?

Algorithme:

1 *ontleed BREUK in GEHELEN, TELLER en NOEMER*
2 *bepaal TELLER/NOEMER en rond deze uitkomst naar beneden af*
3 *tel deze uitkomst bij GEHELEN op*
4 *verminder TELLER met deze uitkomst maal NOEMER*
5 *stel DELER op 2*
6 *is TELLER/DELER een geheel getal? Ga naar punt 10 wanneer dat zo is*
7 *wanneer DELER gelijk is aan 2, stel DELER dan gelijk aan 3. Verhoog DELER anders met 2*
8 *is DELER groter dan TELLER? Ga naar punt 13 wanneer dit zo is*
9 *ga door met punt 6*
10 *is NOEMER/DELER een geheel getal? Ga naar punt 7 wanneer dit niet zo is*
11 *deel TELLER en NOEMER door DELER*
12 *ga naar punt 8*
13 *de vereenvoudigde breuk bestaat uit de delen GEHELEN, TELLER en NOEMER*
14 *einde berekening*

Dit algorithme laat zich uiteindelijk vrij gemakkelijk in het volgende MSX-basic-programma vertalen:

```
10 REM ****************************
20 REM * BREUKEN VEREENVOUDIGEN *
30 REM ****************************
40 REM
50 CLS:PRINT "VEREENVOUDIGEN BREUKEN":I$
="0002TE VEREENVOUDIGEN BREUK: ":GOSUB 1
30:IF E THEN STOP ELSE IF B3=0 THEN 90
60 P=INT(B2/B3):B1=B1+P:B2=B2-P*B3:P=2
70 IF B2<P THEN GOTO 90 ELSE IF B2/P=INT
(B2/P) AND B3/P=INT(B3/P) THEN B2=B2/P:B
3=B3/P:GOTO 70
80 P=P+2+(P=2):GOTO 70
90 PRINT:PRINT B0$;" =";:IF B1 THEN PRIN
T STR$(B1);ELSE IF B2*B3=0 THEN PRINT "
0";
100 IF B2 THEN PRINT STR$(B2);"/";MID$(S
TR$(B3),2) ELSE PRINT
110 PRINT:PRINT "GEEF EEN TOETS IN"
120 IF INKEY$="" THEN 120 ELSE RUN
130 LOCATE VAL(LEFT$(I$,2)),VAL(MID$(I$,
3,2)):PRINT MID$(I$,5);
140 B1=0:B2=0:B3=0:E=0:LINE INPUT B$:B0$
=B$:IF B$="" THEN E=1:RETURN ELSE FOR II
=1 TO LEN(B$):BB$=MID$(B$,II,1):IF (BB$<
"0" OR BB$>"9") AND BB$<>"/" AND BB$<>"
" THEN 130 ELSE NEXT II
150 P=INSTR(B$," "):IF P>1 THEN B1=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1) ELSE IF P
THEN 130 ELSE IF INSTR (B$,"/")=0 THEN B
1=VAL(B$):RETURN
160 P=INSTR(B$,"/"):IF P>1 THEN B2=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1):IF LEN(B$)
 THEN B3=VAL(B$):GOTO 170 ELSE 130 ELSE
```

```
130
170 IF B2<>0 AND B3=0 THEN 130 ELSE RETUR
N
```

4.4 Breuken vermenigvuldigen

Nu we eenmaal een programma hebben ontworpen om breuken te vereenvoudigen, is de rest kinderspel. Wanneer we twee breuken met elkaar willen vermenigvuldigen, behoeven we alleen betreffende breuken nog maar te laten ingeven, de vermenigvuldiging uit te voeren, het resultaat te vereenvoudigen en de uitkomst af te drukken. Alleen de daadwerkelijke breukvermenigvuldiging is nieuw.

Voordat we het vermenigvuldigen van breuken in een programma gaan gieten, dienen we ons eerst te realiseren wat het vermenigvuldigen van breuken nu eigenlijk inhoudt.

Wanneer we de akties voor onszelf op papier zouden moeten zetten, zou het 'recept' voor het vermenigvuldigen van twee breuken er ongeveer als volgt uitzien:

Hoe vermenigvuldig ik twee breuken?

Algemeen:

Maak van elk van de twee breuken één breuk door de gehelen maal de noemer bij de teller te tellen.
Bijvoorbeeld: 3 1/7 is gelijk aan $\frac{3 \times 7 + 1}{7}$ *is gelijk aan 22/7*

Nadat op deze wijze twee breuken zijn verkregen, de tellers met elkaar vermenigvuldigen en de noemers met elkaar vermenigvuldigen.

De uitkomst eventueel weer vereenvoudigen.

Voor de computer maken we een recept dat er wat formeler uitziet: het algorithme:

Hoe vermenigvuldig ik twee breuken?

Algorithme:

1 ontleed BREUK-1 in GEHELEN-1, TELLER-1 en NOEMER-1
2 ontleed BREUK-2 in GEHELEN-2, TELLER-2 en NOEMER-2
3 vermenigvuldig GEHELEN-1 met NOEMER-1 en tel het resultaat op bij TELLER-1
4 vermenigvuldig GEHELEN-2 met NOEMER-2 en tel het resultaat op bij TELLER-2
5 vermenigvuldig TELLER-1 met TELLER-2 en noem het resultaat TELLER
6 vermenigvuldig NOEMER-1 met NOEMER-2 en noem het resultaat NOEMER
7 UITKOMST van de vermenigvuldiging is gelijk aan TELLER/NOEMER
8 vereenvoudig de breuk UITKOMST

Dit algorithme laat zich eenvoudig vertalen in een MSX-programma:

```
10 REM *****************************
20 REM * VERMENIGVULDIGEN BREUKEN *
30 REM *****************************
40 REM
50 CLS:PRINT "VERMENIGVULDIGEN BREUKEN":
PRINT
60 I$="0002GEEF EERSTE BREUK IN: ":GOSUB
170:IF E THEN STOP ELSE C0$=B0$:C1=B1:C2
=B2:C3=B3
70 I$="0003GEEF TWEEDE BREUK IN: ":GOSUB
170:IF E THEN 60
80 B3=B3-(B3=0):C3=C3-(C3=0):B2=B2+B3*B1
:C2=C2+C3*C1:B2=B2*C2:B3=B3*C3:B1=0
90 GOSUB 140
100 PRINT:PRINT C0$;" x ";B0$;" =";:IF B
1 THEN PRINT STR$(B1);ELSE IF B2*B3=0 TH
EN PRINT " 0";
110 IF B2 THEN PRINT STR$(B2);"/";MID$(S
TR$(B3),2) ELSE PRINT
120 PRINT:PRINT "GEEF EEN TOETS IN"
130 IF INKEY$="" THEN 130 ELSE RUN
140 P=INT(B2/B3):B1=B1+P:B2=B2-P*B3:P=2
```

```
ELSE RETURN
150 IF B2<P THEN RETURN ELSE IF B2/P=INT
(B2/P) AND B3/P=INT(B3/P) THEN B2=B2/P:B
3=B3/P:GOTO 150
160 P=P+2+(P=1):GOTO 150
170 LOCATE VAL(LEFT$(I$,2)),VAL(MID$(I$,
3,2)):PRINT MID$(I$,5);
180 B1=0:B2=0:B3=0:E=0:LINE INPUT B$:B0$
=B$:IF B$="" THEN E=1:RETURN ELSE FOR II
=1 TO LEN(B$):BB$=MID$(B$,II,1):IF (BB$<
"0" OR BB$>"9") AND BB$<>"/" AND BB$<>"
" THEN 170 ELSE NEXT II
190 P=INSTR(B$," "):IF P>1 THEN B1=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1) ELSE IF P
THEN 170 ELSE IF INSTR(B$,"/")=0 THEN B1
=VAL(B$):RETURN
200 P=INSTR(B$,"/"):IF P>1 THEN B2=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1):IF LEN(B$)
 THEN B3=VAL(B$):GOTO 210 ELSE 170 ELSE
170
210 IF B2>0 AND B3=0 THEN 170 ELSE RETUR
N
```

Merk op dat dit (en de volgende) programma(s) bijzonder veel lijkt (lijken) op het programma VEREENVOUDIGEN BREUKEN.

Eerst worden twee breuken ingegeven. De ingegeven breuken zijn terug te vinden in de variabelen C1, C2 en C3 (eerste breuk) en B1, B2 en B3 (tweede breuk).

Met deze zes getallen vindt de uiteindelijke breukberekening plaats waarna het eindresultaat wordt vereenvoudigd (als in het vorige programma) en afgedrukt.

4.5 Breuken delen

Het delen van breuken en het vermenigvuldigen van breuken zijn twee akties die erg veel op elkaar lijken.

Allereerst vragen we ons af wat het delen van breuken nu eigenlijk inhoudt:

Hoe deel ik twee breuken?

Algemeen:

Maak weer van elk van de twee breuken één breuk. Draai vervolgens de tweede breuk om en vermenigvuldig de breuken vervolgens.

De uitkomst eventueel weer vereenvoudigen.

Daarna ontwerpen we een strak en foutloos algorithme:

Hoe deel ik twee breuken?

Algorithme:

1 *ontleed BREUK-1 in GEHELEN-1, TELLER-1 en NOEMER-1*
2 *ontleed BREUK-2 in GEHELEN-2, TELLER-2 en NOEMER-2*
3 *vermenigvuldig GEHELEN-1 met NOEMER-1 en tel het resultaat op bij TELLER-1*
4 *vermenigvuldig GEHELEN-2 met NOEMER-2 en tel het resultaat op bij TELLER-2*
5 *vermenigvuldig TELLER-1 met NOEMER-2 en noem het resultaat TELLER*
6 *vermenigvuldig TELLER-2 met NOEMER-2 en noem het resultaat NOEMER*
7 *UITKOMST van de deling is gelijk aan TELLER/NOEMER*
8 *vereenvoudig de breuk UITKOMST*

Uiteindelijk maken we met behulp van dit algorithme een MSX-basic-programma voor het delen van breuken:

```
10 REM ******************************
20 REM *        DELEN BREUKEN       *
30 REM ******************************
40 REM
50 CLS:PRINT "DELEN BREUKEN":PRINT
60 I$="0002GEEF EERSTE BREUK IN: ":GOSUB
170:IF E THEN STOP ELSE C0$=B0$:C1=B1:C2
```

```
=B2:C3=B3
70 I$="0003GEEF TWEEDE BREUK IN: ":GOSUB
170:IF E THEN 60
80 B3=B3-(B3=0):C3=C3-(C3=0):B2=B2+B3*B1
:C2=C2+C3*C1:Q=B2:B2=C2*B3:B3=C3*Q:B1=0
90 GOSUB 140
100 PRINT:PRINT C0$;" / ";B0$;" =";:IF B
1 THEN PRINT STR$(B1);ELSE IF B2*B3=0 TH
EN PRINT " 0";
110 IF B2 THEN PRINT STR$(B2);"/";MID$(S
TR$(B3),2) ELSE PRINT
120 PRINT:PRINT "GEEF EEN TOETS IN"
130 IF INKEY$="" THEN 130 ELSE RUN
140 P=INT(B2/B3):B1=B1+P:B2=B2-P*B3:P=2
ELSE RETURN
150 IF B2<P THEN RETURN ELSE IF B2/P=INT
(B2/P) AND B3/P=INT(B3/P) THEN B2=B2/P:B
3=B3/P:GOTO 150
160 P=P+2+(P=1):GOTO 150
170 LOCATE VAL(LEFT$(I$,2)),VAL(MID$(I$,
3,2)):PRINT MID$(I$,5);
180 B1=0:B2=0:B3=0:E=0:LINE INPUT B$:B0$
=B$:IF B$="" THEN E=1:RETURN ELSE FOR II
=1 TO LEN(B$):BB$=MID$(B$,II,1):IF (BB$<
"0" OR BB$>"9") AND BB$<>"/" AND BB$<>"
" THEN 170 ELSE NEXT II
190 P=INSTR(B$," "):IF P>1 THEN B1=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1) ELSE IF P
THEN 170 ELSE IF INSTR(B$,"/")=0 THEN B1
=VAL(B$):RETURN
200 P=INSTR(B$,"/"):IF P>1 THEN B2=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1):IF LEN(B$)
 THEN B3=VAL(B$):GOTO 210 ELSE 170 ELSE
170
210 IF B2>0 AND B3=0 THEN 170 ELSE RETUR
N
```

4.6 Optellen van breuken

Volgens de inmiddels bekende weg vragen we ons eerst af wat het optellen van breuken nu eigenlijk inhoudt:

Hoe tel ik twee breuken op?

Algemeen:

Maak weer van elk van de twee breuken één breuk. Vermenigvuldig van elk van de breuken de teller met de noemer van de andere. Tel deze twee tellers daarna bij elkaar op. Vermenigvuldig uiteindelijk de beide noemers met elkaar.

Vereenvoudig eventueel de uitkomst.

...waarna we weer het bouwplan van ons uiteindelijk programma ontwerpen:

Hoe tel ik twee breuken bij elkaar op?

Algorithme:

1 *ontleed BREUK-1 in GEHELEN-1, TELLER-1 en NOEMER-1*
2 *ontleed BREUK-2 in GEHELEN-2, TELLER-2 en NOEMER-2*
3 *vermenigvuldig NOEMER-1 met GEHELEN-1 en tel het resultaat op bij TELLER-1*
4 *vermenigvuldig NOEMER-2 met GEHELEN-2 en tel het resultaat op bij TELLER-2*
5 *stel TELLER-1 gelijk aan TELLER-1 maal NOEMER-2*
6 *stel TELLER-2 gelijk aan TELLER-2 maal NOEMER-1*
7 *stel TELLER gelijk aan TELLER-1 plus TELLER-2*
8 *stel NOEMER gelijk aan NOEMER-1 maal NOEMER-2*
9 *UITKOMST van de optelling is TELLER/NOEMER*
10 *vereenvoudig deze breuk*

...om uiteindelijk tot het MSX-basic-programma te komen dat de breuken perfekt voor ons bij elkaar optelt.

```
10 REM ****************************
20 REM *     OPTELLEN BREUKEN      *
30 REM ****************************
40 REM
50 CLS:PRINT "OPTELLEN BREUKEN":PRINT
60 I$="0002GEEF EERSTE BREUK IN: ":GOSUB
170:IF E THEN STOP ELSE C0$=B0$:C1=B1:C2
=B2:C3=B3
70 I$="0003GEEF TWEEDE BREUK IN: ":GOSUB
170:IF E THEN 60
80 B3=B3-(B3=0):C3=C3-(C3=0):B2=B2+B3*B1
:C2=C2+C3*C1:B2=C2*B3+C3*B2:B3=B3*C3:B1=
0
90 GOSUB 140
100 PRINT:PRINT C0$;" + ";B0$;" =";:IF B
1 THEN PRINT STR$(B1);ELSE IF B2*B3=0 TH
EN PRINT " 0";
110 IF B2 THEN PRINT STR$(B2);"/";MID$(S
TR$(B3),2) ELSE PRINT
120 PRINT:PRINT "GEEF EEN TOETS IN"
130 IF INKEY$="" THEN 130 ELSE RUN
140 P=INT(B2/B3):B1=B1+P:B2=B2-P*B3:P=2
ELSE RETURN
150 IF B2<P THEN RETURN ELSE IF B2/P=INT
(B2/P) AND B3/P=INT(B3/P) THEN B2=B2/P:B
3=B3/P:GOTO 150
160 P=P+2+(P=2):GOTO 150
170 LOCATE VAL(LEFT$(I$,2)),VAL(MID$(I$,
3,2)):PRINT MID$(I$,5);
180 B1=0:B2=0:B3=0:E=0:LINE INPUT B$:B0$
=B$:IF B$="" THEN E=1:RETURN ELSE FOR II
=1 TO LEN(B$):BB$=MID$(B$,II,1):IF (BB$<
"0" OR BB$>"9") AND BB$<>"/" AND BB$<>"
" THEN 170 ELSE NEXT II
190 P=INSTR(B$," "):IF P>1 THEN B1=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1) ELSE IF P
```

```
THEN 170 ELSE IF INSTR(B$,"/")=0 THEN B1
=VAL(B$):RETURN
200 P=INSTR(B$,"/"):IF P>1 THEN B2=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1):IF LEN(B$)
 THEN B3=VAL(B$):GOTO 210 ELSE 170 ELSE
170
210 IF B2>0 AND B3=0 THEN 170 ELSE RETUR
N
```

4.7 Aftrekken van breuken

Om het verhaal 'rekenen met breuken' kompleet te maken, moeten we ook breuken van elkaar kunnen aftrekken.

De algemene formulering:

Hoe trek ik twee breuken van elkaar af?

Algemeen:

Als het optellen van twee breuken. Trek in dit geval echter de twee tellers na vermenigvuldiging van elkaar af.

Het bouwplan:

Hoe trek ik twee breuken van elkaar af?

Algorithme:

1 *ontleed BREUK-1 in GEHELEN-1, TELLER-1 en NOEMER-1*
2 *ontleed BREUK-2 in GEHELEN-2, TELLER-2 en NOEMER-2*
3 *vermenigvuldig NOEMER-1 met GEHELEN-1 en tel het resultaat op bij TELLER-2*
4 *vermenigvuldig NOEMER-2 met GEHELEN-2 en tel het resultaat op bij TELLER-2*
5 *stel TELLER-1 gelijk aan TELLER-1 maal NOEMER-2*
6 *stel TELLER-2 gelijk aan TELLER-2 maal NOEMER-1*
7 *stel TELLER gelijk aan TELLER-1 min TELLER-2*
8 *stel NOEMER gelijk aan NOEMER-1 maal NOEMER-2*

9 *UITKOMST van de aftrekking is TELLER/NOEMER. Pas op, deze uitkomst kan negatief zijn!*
10 *vereenvoudig deze breuk*

...en uiteindelijk ons programma. Eén extra moeilijkheid doet zich voor. Bij het aftrekken van breuken kan een negatieve breuk ontstaan. Hiermee is in het onderstaande programma rekening gehouden.

```
10 REM *****************************
20 REM *    AFTREKKEN BREUKEN      *
30 REM *****************************
40 REM
50 CLS:PRINT "AFTREKKEN BREUKEN":PRINT
60 I$="0002GEEF EERSTE BREUK IN: ":GOSUB
170:IF E THEN STOP ELSE C0$=B0$:C1=B1:C2
=B2:C3=B3
70 I$="0003GEEF TWEEDE BREUK IN: ":GOSUB
170:IF E THEN 60
80 M=0:B3=B3-(B3=0):C3=C3-(C3=0):B2=B2+B
3*B1:C2=C2+C3*C1:B2=C2*B3-C3*B2:B3=B3*C3
:B1=0:IF B2<0 THEN M=1:B2=-B2
90 GOSUB 140
100 PRINT:PRINT C0$;" - ";B0$;" = ";STRI
NG$(M,"-");:IF B1 THEN PRINT MID$(STR$(B
1),2);" ";ELSE IF B2*B3=0 THEN PRINT "0"
;
110 IF B2 THEN PRINT MID$(STR$(B2),2);"/
";MID$(STR$(B3),2) ELSE PRINT
120 PRINT:PRINT "GEEF EEN TOETS IN"
130 IF INKEY$="" THEN 130 ELSE RUN
140 P=INT(B2/B3):B1=B1+P:B2=B2-P*B3:P=2
ELSE RETURN
150 IF B2<P THEN RETURN ELSE IF B2/P=INT
(B2/P) AND B3/P=INT(B3/P) THEN B2=B2/P:B
3=B3/P:GOTO 150
160 P=P+2+(P=2):GOTO 150
170 LOCATE VAL(LEFT$(I$,2)),VAL(MID$(I$,
```

```
3,2)):PRINT MID$(I$,5);
180 B1=0:B2=0:B3=0:E=0:LINE INPUT B$:B0$
=B$:IF B$="" THEN E=1:RETURN ELSE FOR II
=1 TO LEN(B$):BB$=MID$(B$,II,1):IF (BB$<
"0" OR BB$>"9") AND BB$<>"/" AND BB$<>"
" THEN 170 ELSE NEXT II
190 P=INSTR(B$," "):IF P>1 THEN B1=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1) ELSE IF P
THEN 170 ELSE IF INSTR(B$,"/")=0 THEN B1
=VAL(B$):RETURN
200 P=INSTR(B$,"/"):IF P>1 THEN B2=VAL(L
EFT$(B$,P-1)):B$=MID$(B$,P+1):IF LEN(B$)
 THEN B3=VAL(B$):GOTO 210 ELSE 170 ELSE
170
210 IF B2>0 AND B3=0 THEN 170 ELSE RETUR
N
```

5 Twee onbekende MSX bevelen

MSX-basic is een taal in ontwikkeling. Enkele jaren geleden kwam ik in aanraking met wat Toshiba-computers. Pas veel later besefte ik, dat deze Toshiba-computers de voorlopers waren van de MSX-standaard.

Toen ik onlangs het MSX-ROM nog een keer goed doorzocht, ontdekte ik plotseling een zestal nooit gedokumenteerde bevelen. In mijn oude Toshiba-handboeken vond ik echter gedeeltelijk het juiste gebruik van deze bevelen terug!

Twee van deze bevelen zijn wel heel erg praktisch: ze staan toe om hele sektoren tegelijk van schijf te lezen en weer naar schijf terug te schrijven:

Het DSKI$-bevel

DSKI$ is een systeemvariabele en heeft tussen haakjes twee waarden nodig. DSKI$(x,y) geeft van floppy-eenheid nummer x de inhoud van sektor nummer y. Indien x gelijk is aan 0, wordt de laatst aangesproken floppy-eenheid benaderd. Het maximum voor y is per merk floppy verschillend.

Er zit één addertje onder het gras. Wanneer men programmeert:

```
10 LET A$=DSKI$(0,4)
```

Dan zou men verwachten dat A$ gelijk zou worden gesteld aan de inhoud van sektor 4 van de eerste schijf. Niets is echter minder waar.

Wat er wél gebeurt is, dat in een buffer in het systeemgedeelte van de MSX-computer de sektor wordt ingelezen! Het adres van het eerste byte van deze buffer is PEEK(62289) + 256*PEEK(62290). Het is dus

nog vrij ingewikkeld om de inhoud van de betreffende sektor te benaderen!

Het DSKO$-bevel

Het DSKO$-bevel staat toe, de eerder genoemde systeembuffer weer op schijf terug te zetten. DSKO$ x,y schrijft op floppy-eenheid nummer x, sektor nummer y terug (zie DSKI$). Wanneer men programmeert:

```
20 DSKO$ 0,2
```

Dan schrijft men op floppy-eenheid nummer 0 (de eerste eenheid) sektor nummer 2 weg.

Het zal duidelijk zijn dat het gebruik van DSKI$ en DSKO$ vele gevaren met zich meebrengt. Een verkeerd gebruik kan resulteren in een totale verminking van de schijf!

Wanneer u overweegt, deze bevelen toe te passen, probeer het één en ander dan eerst uit op een werkschijfje (dat eventueel verloren mag gaan).

Het DSKI$-bevel en het DSKO$-bevel hebben enkele fraaie toepassingsmogelijkheden. Eén van de mogelijkheden van toepassing is verwerkt in het volgende programma.

Het programma DSP (Disk Surface Processor) dat hieronder volgt, biedt de mogelijkheid om een floppy disk sektor voor sektor te onderzoeken en eventueel te veranderen.

DSP is een heel dankbaar programma wanneer een schijf op blokniveau moet worden veranderd (reparatie van een schijf. Dit moet je natuurlijk wél kunnen...). Maar ook wanneer u de opbouw van een MSX-schijf wilt bestuderen (hoe zit de index in elkaar? Wat staat er in systeemblok 02?) is DSP een erg handig programma.

DSP is eenvoudig te bedienen en omvat de volgende funkties:

G met de G-funktie kan een blok van schijf worden gelezen. Vermeld achter G het nummer van het te lezen blok. G0 leest blok nul in, GA leest blok 10 in, enzovoorts. De bloknummers moeten hexadecimaal worden opgegeven.

P het tegenovergestelde kommando. Met P wordt een eerder gelezen en eventueel veranderd blok weer op schijf teruggeschreven. Vermeld achter P het nummer dat het blok dient te krijgen.

D met de D-funktie kan een blok worden bekeken op inhoud. Achter de D kan eventueel (hexadecimaal) het adres worden opgegeven waarvanaf de inhoud dient te worden gepresenteerd. Maximaal 128 bytes worden achter elkaar geprojekteerd; daarna is een nieuw D-kommando noodzakelijk (alleen ingave D = verder lijsten). De projektie van de inhoud van het blok wordt door intoetsing van een willekeurige toets onmiddellijk beëindigd. Gegevens worden hexadecimaal en in ASCII-kode (leesbaar) geprojekteerd.

Door (hexadecimaal) alleen een adres in te geven, wordt de inhoud van het byte op het ingegeven adres geprojekteerd waarna een andere inhoud kan worden ingegeven. Geef twee karakters in wanneer u een hexadecimale inhoud wenst op te geven en geef slechts één karakter in wanneer u de inhoud in ASCII (direkt leesbare karakters) wilt opgeven. Alleen een RETURN laat de inhoud ongewijzigd. Na een verandering wordt onmiddellijk het volgende adres in behandeling genomen. Ook alleen een RETURN-toets zorgt ervoor dat het direkt volgende adres in behandeling wordt genomen.

```
10 REM **************************
20 REM *          DSP           *
30 REM *          ---           *
40 REM * (C) 1985 STARK TEXEL   *
50 REM **************************
60 REM
70 REM **************************
80 REM * BEPAAL BLOKGROOTTE,    *
90 REM * AANTAL BLOKKEN EN      *
100 REM* DISK-BUFFER            *
110 REM**************************
120 REM
130 CLEAR 10000:P1=PEEK(62289!)+256*PEEK
(62290!):A$="":P2=VARPTR(A$):POKE P2,128
:POKE P2+1,P1-256*(INT(P1/256)):POKE P2+
2,INT(P1/256)
```

```
140 S$=DSKI$(0,0):BL=ASC(MID$(A$,12))+25
6*ASC(MID$(A$,13))
150 N=ASC(MID$(A$,20))+256*ASC(MID$(A$,2
1))
160 DIM A$(BL/128-1):FOR I=0 TO BL/128-1
:A$(I)="":P2=VARPTR(A$(I)):POKE P2,128:P
OKE P2+1,P1-256*(INT(P1/256)):POKE P2+2,
INT(P1/256):P1=P1+128:NEXT I
170 REM
180 REM **************************
190 REM *    HOOFDPROGRAMMA      *
200 REM **************************
210 REM
220 DIM Q$(1):CLS:WIDTH 40:C$="GgPpDd":H
$="0123456789ABCDEF0123456789abcdef"
230 LINE INPUT K$:IF K$="" THEN K$=" "
240 B$=LEFT$(K$,1):B=INSTR(C$,B$):IF B=0
 THEN LOCATE 0,CSRLIN-1:GOSUB 610:GOTO 2
30
250 B=(B-1)/2+1:ON B GOSUB 310,400,500
260 GOTO 230
270 REM **************************
280 REM *    OPHALEN BLOK        *
290 REM **************************
300 REM
310 F$=MID$(K$,2):IF F$="" THEN BEEP:RET
URN
320 GOSUB 830:IF F=-1 THEN BEEP:RETURN
330 IF F>=N THEN BEEP:RETURN ELSE Q$=DSK
I$(0,F)
340 ERASE Q$:DIM Q$(BL/128)
350 FOR I=0 TO BL/128-1:Q$(I+1)=A$(I):NE
XT I:RETURN
360 REM **************************
370 REM *   SCHRIJVEN BLOK       *
380 REM **************************
```

```
390 REM
400 IF Q$(1)="" THEN RETURN
410 F$=MID$(K$,2):IF F$="" THEN BEEP:RET
URN
420 GOSUB 830:IF F=-1 THEN BEEP:RETURN
430 IF F>=N THEN BEEP:RETURN
440 FOR I=0 TO BL/128-1:LSET A$(I)=Q$(I+
1):NEXT I
450 DSKO$ 0,F:RETURN
460 REM ***************************
470 REM *    DISPLAY GEGEVENS    *
480 REM ***************************
490 REM
500 IF Q$(1)="" THEN RETURN
510 Z$="":F$=MID$(K$,2):GOSUB 830:IF F=-
1 THEN RETURN ELSE IF F$>"" THEN D=F
520 FOR J=1 TO 16:Q=D:GOSUB 760:PRINT RI
GHT$(U$,4);":";:FOR I=1 TO 8:B1=D\128:B2
=D-128*B1:B1=B1+1:B2=B2+1
530 IF D>=BL THEN D=0:PRINT:RETURN
540 K=ASC(MID$(Q$(B1),B2)):IF K>31 AND K
<128 THEN Z$=Z$+CHR$(K) ELSE Z$=Z$+"."
550 Q=K:GOSUB 760:PRINT RIGHT$(U$,2); "
";:D=D+1:IF INKEY$>"" THEN PRINT: RETURN
 ELSE NEXT I:PRINT " ";Z$:Z$="":NEXT J:R
ETURN
560 B=B-8
570 REM ***************************
580 REM *   VERANDER GEGEVENS    *
590 REM ***************************
600 REM
610 IF Q$(1)="" THEN RETURN
620 IF K$=" " THEN K$=""
630 F$=K$:GOSUB 830:IF F=-1 THEN BEEP:RE
TURN ELSE IF F$>"" THEN D=F ELSE D=D+1
```

```
640 B1=D\128:B2=D-128*B1:B1=B1+1:B2=B2+1
650 IF D>=BL THEN RETURN
660 Q=D:GOSUB 760:PRINT RIGHT$(U$,4);":"
;
670 K=ASC(MID$(Q$(B1),B2)):IF K>31 AND K
<128 THEN Z$=CHR$(K) ELSE Z$="."
680 Q=K:GOSUB 760:PRINT RIGHT$(U$,2);" "
;Z$;":";
690 LINE INPUT F$:IF F$="" THEN RETURN
700 IF LEN(F$)=1 THEN K=ASC(F$) ELSE GOS
UB 830:K=F:IF K>255 OR K<0 THEN BEEP:GOT
O 640
710 MID$(Q$(B1),B2,1)=CHR$(K):D=D+1:GOTO
 640
720 REM ***************************
730 REM *      Q HEX NAAR U$        *
740 REM ***************************
750 REM
760 U$="00000000"
770 IF Q>15 THEN Q=Q/16:GOSUB 770
780 U$=U$+MID$(H$,Q+1,1):Q=(Q-INT(Q))*16
:RETURN
790 REM ***************************
800 REM *       F$ DEC NAAR F       *
810 REM ***************************
820 REM
830 F=0:IF F$="" THEN RETURN ELSE FOR K=
1 TO LEN(F$):W=INSTR(H$,MID$(F$,K,1)):IF
 W=0 THEN F=-1:RETURN ELSE IF W>16 THEN
W=W-16
840 F=F*16+W-1:NEXT K:RETURN
```

6 Beveiliging

Naarmate u verder vordert in MSX, zal steeds vaker het probleem 'beveiliging' om de hoek komen kijken. Uw programma's worden immers steeds mooier en steeds vaker zult u deze programma's tegen onbevoegd kopiëren willen beschermen. Of misschien maakt u programma's voor derden en moeten bepaalde delen van het programma voor slechts een beperkt aantal mensen toegankelijk zijn.

Beveiliging is een term die een brede lading dekt. Op verschillende niveaus kunnen steeds andere beveiligingen gewenst zijn. Wanneer u als onderwijzer(es) een trainingsprogramma hebt geschreven, dan is het niet de bedoeling dat elke leerling de klasresultaten kan opvragen. Deze opvraging moet dus door bijvoorbeeld een kodewoord zijn beveiligd.

De eerste manier van beveiliging die we hier behandelen, is de zogenaamde PERMISSIEBEVEILIGING.

De permissiebeveiliging gaat er van uit dat bepaalde delen van programmatuur slechts voor een beperkt aantal mensen toegankelijk zijn. De delen die niet voor iedereen toegankelijk zijn, worden meestal door middel van een kodewoord 'afgezekerd.'

De onderstaande routine kan als subroutine in een programma worden opgenomen. Door eerst de variabele I$ te vullen met "XXYYK---K" (XX/YY = positie horizontaal/vertikaal op beeldscherm, K---K = het in te geven kodewoord) en vervolgens deze routine met een GOSUB aan te roepen, bewerkstelligt u de ingave van het kodewoord. Deze routine wordt alleen met het juiste kodewoord verlaten; de 'RETURN' is dus altijd een goed kodewoord.
Dit betekent dat degene die zich in het verkeerde programma (-onderdeel) heeft gewaagd, altijd de hulp zal moeten inroepen van degene

die het kodewoord kent. Dit zal de 'illegale' gebruiker ervan weerhouden, een tweede keer te proberen om in het verboden programma (-onderdeel) te komen!

Na de kodewoordingave, waarbij uiteraard het kodewoord niet op beeldscherm wordt afgebeeld maar er slechts kruisjes worden geprojekteerd, dient de goede programmeur dus nog een akkoord-afvraag te programmeren.

```
10 REM ***************************
20 REM *      KODEWOORDINGAVE    *
30 REM *                         *
40 REM * I$="XXYYK---K"          *
50 REM *                         *
60 REM * XX/YY=POS. HOR/VERT     *
70 REM * K---K=IN TE GEVEN KODE *
80 REM ***************************
90 REM
10000 PW$="":LOCATE VAL(LEFT$(I$,2)),VAL
(MID$(I$,3,2)),1:PRINT "KODEWOORD:";
10010 K$=INKEY$:IF K$="" THEN 10010
10020 IF K$<>CHR$(8) AND (ASC(K$)<32 OR
ASC(K$)>127) THEN BEEP:GOTO 10010 ELSE I
F K$<>CHR$(8) THEN PRINT "X";:PW$=PW$+K$
:IF PW$=MID$(I$,5) THEN PLAY "M9999S1T32
L16CDE":RETURN ELSE 10010
10030 IF PW$>"" THEN PW$=LEFT$(PW$,LEN(P
W$)-1):PRINT K$;" ";K$;:GOTO 10010 ELSE
BEEP:GOTO 10010
```

De tweede manier van beveiliging die we hier behandelen, is de MISLEIDENDE BEVEILIGING.

De misleiding wordt toegepast wanneer we een persoonlijk programma willen afschermen tegen gebruik door derden. Door de 'illegale' gebruiker te misleiden, zetten we hem of haar op het verkeerde spoor waardoor het daadwerkelijke programma ongebruikt blijft.

De onderstaande routine kan vooraan in het programma worden opge-

nomen. De routine simuleert een normale MSX-computerstart. De argeloze gebruiker zal denken dat de basic-editor gewoon aktief is. Echter, op alles wat hij of zij intoetst, antwoordt het programma met een Syntax error. Na verwoede list-pogingen geeft de programma-inbreker het uiteindelijk wellicht op.

U, die bekend bent met deze misleidende beveiliging, geeft slechts het woordje 'kode' in (of een ander kodewoord nadat u het programma hebt aangepast) waarna het werkelijke programma aktief wordt!

De grote kracht van een dergelijke beveiliging is, dat degene waartegen wordt beveiligd meestal niet beseft dat hij of zij hier met een beveiliging van doen heeft...

```
1 ONSTOPGOSUB4:STOPON
2 KEYON:SCREEN0:WIDTH37:COLOR15,4,4:CLS:
PRINT"MSX BASIC version 1.0":PRINT"Copyr
ight 1983 by Microsoft":PRINT"24455 Byte
s free":PRINT"Disk BASIC version 1.0"
3 LINEINPUTA$:IFA$=""THEN3ELSEIFA$<>"kod
e"THENPRINT"Syntax error":BEEP:PRINT"Ok"
:GOTO3ELSE6
4 RETURN3
5 REM
6 REM ***** PROGRAMMA *****
```

De derde manier van beveiliging die we hier behandelen, is de BREEK-BEVEILIGING.

De bedoeling van deze BREEK-beveiliging is dat het programma door derden wel mag worden GEBRUIKT maar niet mag worden BESTUDEERD of ONDERBROKEN. Allereerst moet het programma dus ongevoelig worden gemaakt voor gebruik van de CONTROL-BREAK-optie. Wanneer het de 'inbreker' toch lukt om het programma stil te leggen, dan dient het programma niet te kunnen worden geLIST.

Het onderstaande programma kan vooraan in het te beveiligen programma worden opgenomen en is een heel 'gemeen' beveiligingsprogramma. Veel truuks die de programma-kraker zou kunnen aanwenden om door te dringen in de programma-opbouw, worden door deze beveiliging geëlimineerd. Alleen de 'meester-kraker' zal op de duur

sukses hebben.

Gaat u bij beveiliging als volgt te werk:

1) Zorg dat het nog onbeveiligde programma vlekkeloos werkt en dat ERRORS met ON ERROR voldoende worden opgevangen. Met ON STOP dient u de CONTROL-STOP op juiste wijze te hebben afgevangen.

2) Voeg het onderstaande programma VOORAAN toe aan uw programma in. Pas op: tik alles tot op de spatie nauwkeurig in. Elke tikfout resulteert er later in, dat u helemaal van voor af aan moet beginnen!

3) Ook u kunt het programma straks niet meer kraken! SAVE de nog niet beveiligde versie dus EERST!

4) Geef een RUN. Geef na de Ok-melding een LIST. Alleen de eerste regel is nog maar zichtbaar en is totaal nietszeggend.

5) SAVE de nu beveiligde versie onder een ANDERE naam.

In de onderstaande beveiligingsroutine zitten POKE-bevelen die er onder meer voor zorgdragen dat het programma onzichtbaar wordt en dat het programma, mocht het worden onderbroken, niet LISTbaar is. Ook aktiveren zij een eventueel door een heel slimme kraker gedeaktiveerde CONTROL-STOP weer.

Wat de POKEs precies voorstellen, zullen we hier maar onbesproken laten. De potentiële kraker leest mee...

```
1 POKE32840!,255:POKE32841!,255:DELETE1
2 ONSTOPGOSUB2:STOPON:ONERRORGOTO2:READA
:IFATHENREADB:POKEA,B:GOTO2
3 DATA 64922,201,64923,201,64924,201,649
25,201,65417,199,32811,0,32812,0,0
4 REM
5 REM ***** PROGRAMMA *****
```

Voor de amateur met een 16 Kb MSX-computer: tel de waarde 16384 op bij de waarden 32840, 32841, 32811 en 32812 in dit programma.

Alle volgende beveiligingsmanieren zijn alleen van toepassing op een MSX-computer met een (echte) floppy disk eenheid (dus géén Quick Disk).

De vierde manier van beveiliging die we hier behandelen, is de MEDIUM-BEVEILIGING.

De medium-beveiliging beschermt de gebruiker tegen het plaatsen van verkeerde media (schijven). Deze beveiliging is noodzakelijk wanneer we professionele software gaan samenstellen waarbij we er vanuit gaan dat de benodigde gegevens op verschillende schijven staan. Om een goede werking van deze software te kunnen waarborgen, dienen we programmatechnisch te kunnen kontroleren of de juiste schijf wel in het station is geplaatst.

Met het volgende programma is het mogelijk om een schijf te voorzien van een maximaal 16 cijferig schijfnummer. Dit schijfnummer hoort bij de schijf zelf en wordt NOOIT meegekopiëerd. Alleen het initialiseren van een schijf heeft tot gevolg dat het schijfnummer vervalt.

Het schijfnummer wordt in het eerste op schijf aanwezige blok opgenomen en wel op een plaats die op dit moment door MSX nog niet wordt gebruikt. Het schijfnummer is volkomen onzichtbaar aanwezig op de schijf.

```
10 REM ***************************
20 REM * INBRENGEN SERIENUMMER *
30 REM * VOOR EEN FLOPPY DISK   *
40 REM ***************************
50 REM
60 CLS:PRINT "NUMMEREN FLOPPY DISKS":PRI
NT
70 SA=PEEK(62289!)+256*PEEK(62290!)+240
80 S$=DSKI$(0,0):S$="":IF PEEK(SA)=0 THE
N PRINT "FLOPPY HEEFT NOG GEEN SERIENUMM
ER.":GOTO 100
90 FOR I=SA TO SA+15:S$=S$+CHR$(PEEK(I))
:NEXT I
100 LOCATE 0,4:PRINT "SERIENUMMER:"+S$+S
PACE$(40):LOCATE 12,4:LINE INPUT S$
```

```
110 IF LEN(S$)>16 THEN S$="":BEEP:GOTO 1
00 ELSE S$=LEFT$(S$+SPACE$(16),16)
120 FOR I=SA TO SA+15:POKE I,ASC(MID$(S$
,I-SA+1)):NEXT I:DSKO$ 0,0
130 PRINT:PRINT "FLOPPY IS GENUMMERD!":S
TOP
```

Het volgende, éénregelige routinetje haalt het schijfnummer op en plaatst het in S$. Wanneer op een bepaald punt in het programma moet worden gekontroleerd of de juiste schijf is geplaatst, kan het nummer van de schijf met deze regel dus worden opgehaald waarna het kan worden gekontroleerd.

```
1 I=PEEK(62289!)+256*PEEK(62290!)+240:S$
=DSKI$(0,0):S$="":FOR I=I TO I+15:S$=S$+
CHR$(PEEK(I)):NEXT I
2 REM
3 REM ***** PROGRAMMA *****
```

De vijfde en laatste manier van beveiliging die we hier gaan behandelen, is de KOPIEER-BEVEILIGING.

Met de kopiëer-beveiliging willen we bewerkstelligen dat:

- het programma wél op onze computer kan worden gebruikt
- het programma echter NIET onderbreekbaar is
- het programma NIET te LISTen is
- het programma na kopiëren NIET MEER BRUIKBAAR IS!!!

Behalve dat het programma (zie de BREEK-beveiliging) niet meer te 'kraken' is, is het nu ook niet meer illegaal te kopiëren.

De slimme lezer zal opmerken dat een vorm van kopiëer-beveiliging al in het vorige gedeelte werd gegeven. Wanneer men een schijf een bepaald (onzichtbaar) serienummer meegeeft en wanneer men daarbij op cryptische wijze op verschillende plaatsen in het te beveiligen programma dit serienummer op juistheid afvraagt en wanneer men daarbij dan ook nog het programma de eerder behandelde breek-beveiliging meegeeft, dan is het programma toch wel erg goed beveiligd! Het grappige is dat het programma wel naar een andere schijf is te kopiëren maar dat het daar later niet zal blijken te funktioneren!

Wanneer men echter beschikt over een MSX floppy disk eenheid met een standaard blokgrootte van minstens 512 bytes (en alle mij tot nu toe bekende floppy-eenheden hebben minimaal deze blokgrootte), dan kan het nog veiliger!

De volgende programma-toevoeging maakt het mogelijk, de kopiëer-beveiliging zelfs voor de meesters der meesterkrakers onoverwinnelijk te maken.

De essentie van deze beveiliging is, dat er 254 bytes uit het te beveiligen programma worden verwijderd en in het systeemblok 0 van de schijf worden geplaatst. Het programma dat eventueel wordt gekopiëerd, heeft dus gewoon een groot 'gat' in zich en is zelfs door de grootste bollebozen niet meer te repareren. Alleen wanneer het beveiligde programma op de juiste schijf wordt geRUNd, wordt het 'gat' in het programma vanuit systeemblok 0 weer opgevuld en is het programma weer in orde.

Doordat het programma zich wel gewoon laat kopiëren, heeft de software-dief in eerste instantie het idee dat zijn diefstal is geslaagd. Pas thuis zal hij of zij er achterkomen, dat er iets aan het programma mankeert. Nog heel veel uren later komt hij of zij er misschien pas achter dat er een gedeelte van het programma gewoon niet aanwezig is.

Alles wordt nog eens extra moeilijk gemaakt doordat de eerder behandelde breek-beveiliging bij deze beveiliging is opgenomen.

Gaat u bij deze beveiliging als volgt te werk:

1) Zorg dat het nog onbeveiligde programma vlekkeloos werkt en dat ERRORs en onderbrekingen via CONROL-BREAK naar behoren zijn afgevangen.

2) Voeg de onderstaande regels vooraan en achteraan toe aan uw programma. Pas op, tik alles precies in zoals het er staat. Een spatie te veel kan al een fout veroorzaken.

3) Ook u kunt het programma dadelijk niet meer kraken. SAVE daarom de nu nog niet beveiligde versie.

4) geef GOTO 65528 in. Na verloop van tijd verschijnt de foutmelding 'Illegal function call in 65529.' Deze foutmelding hoort zo; niet in paniek raken dus.

5) LIST het programma. Alleen de eerste, nietszeggende regel is nog zichtbaar.

6) SAVE het nu beveiligde programma onder een ANDERE NAAM.

7) RUN het programma en kontroleer de werking.

8) KOPIEER het programma naar een andere schijf (b.v. via een LOAD, gevolgd door een SAVE nadat de schijf is verwisseld) en RUN het daar nog een keer. Het zal nu een fout geven, gewoon stoppen, verkeerd werken of de computer vastzetten. In ieder geval werkt het van geen kant! Een bewijs dat de kopiëerbeveiliging uitmuntend werkt!

9) Beveilig per floppy nooit meer dan één programma. Twee of meer programma's per floppy op deze wijze beveiligen, is helaas onmogelijk!

```
1 ONSTOPGOSUB1:STOPON:ONERRORGOTO1:READA
:IFATHENREADB:POKEA,B:GOTO1
2 DATA 64922,201,64923,201,64924,201,649
25,201,65417,199,32811,0,32812,0,0
3 SA=PEEK(62289!)+256*PEEK(62290!)+256:S
$=DSKI$(0,0):ST=PEEK(SA)+256*PEEK(SA+1):
FOR I=2TO255:POKEST+I-2,PEEK(SA+I):NEXTI
:REM()
7 REM
8 REM ***** PROGRAMMA *****
9 REM
65528 SA=PEEK(62289!)+256*PEEK(62290!)+2
56:S$=DSKI$(0,0):FORI=32771!TOPEEK(63170
!)+256*PEEK(63171!)-1:IFPEEK(I)=40ANDPEE
K(I+1)=41THENST=I+3ELSENEXTI:STOP
65529 POKESA,ST-256*INT(ST/256):POKESA+1
,INT(ST/256):FORI=2TO255:POKESA+I,PEEK(S
T+I-2):POKEST+I-2,255:NEXTI:DSKO$0,0:POK
E32811!,255:POKE32812!,255:DELETE65528-6
5529
```

Voor de amateur met een 16 Kb MSX-computer geldt: tel bij de waarden 32811 en 32812 de waarde 16384 op!

7 De geheimen van het BASIC-programma

De meeste amateurs zullen na verloop van tijd wel weten dat je met de bevelen PEEK en POKE het MSX-geheugen kunt besturen en veranderen. Echter, de meesten zullen het gebruik van deze funkties vermijden omdat de opbouw van het geheugen niet bekend is en een willekeurig gebruik vaak tot vervelende situaties kan leiden.

Toch is het niet moeilijk om deze bevelen goed te gebruiken. Wat je er wel voor moet weten is, hoe het geheugen van een MSX-computer is ingedeeld.

In het volgende hoofdstuk gaan we in op de manier waarop een basic programma in een MSX-computer is gekodeerd.

We gaan steeds uit van een MSX-computer met minimaal 32 kilobytes geheugen. Gebruikers van een MSX-computer met maar 16 kilobytes geheugen moeten bij alle genoemde geheugenadressen de waarde 16384 optellen.

7.1 TOKENS

Om de totale opslagcapaciteit van een basic-programma wat te beperken, zijn de sleutelwoorden van BASIC niet als zodanig in het computergeheugen opgenomen. Zij zijn gekodeerd in zogenaamde TOKENS, afkortingen die in één byte passen. Een PRINT, een INPUT en een DRAW kosten als kommando dus slechts één byte van het computergeheugen.

TOKENS hebben in het computergeheugen altijd een waarde van 128 of hoger, met een maximum van 255. Hieronder volgt een tabel van alle tokens die MSX-basic hanteert:

80:<NVT>	81:END	82:FOR
83:NEXT	84:DATA	85:INPUT
86:DIM	87:READ	88:LET
89:GOTO	8A:RUN	8B:IF
8C:RESTORE	8D:GOSUB	8E:RETURN
8F:REM	90:STOP	91:PRINT
92:CLEAR	93:LIST	94:NEW
95:ON	96:WAIT	97:DEF
98:POKE	99:CONT	9A:CSAVE
9B:CLOAD	9C:OUT	9D:LPRINT
9E:LLIST	9F:CLS	A0:WIDTH
A1:LSE	A2:TRON	A3:TROFF
A4:SWAP	A5:ERASE	A6:ERROR
A7:RESUME	A8:DELETE	A9:AUTO
AA:RENUM	AB:DEFSTR	AC:DEFINT
AD:DEFSNG	AE:DEFDBL	AF:LINE
B0:OPEN	B1:FIELD	B2:GET
B3:PUT	B4:CLOSE	B5:LOAD
B6:MERGE	B7:FILES	B8:LSET
B9:RSET	BA:SAVE	BB:LFILES
BC:CIRCLE	BD:COLOR	BE:DRAW
BF:PAINT	C0:BEEP	C1:PLAY
C2:PSET	C3:PRESET	C4:SOUND
C5:SCREEN	C6:VPOKE	C7:SPRITE
C8:VDP	C9:BASE	CA:CALL
CB:TIME	CC:KEY	CD:MAX
CE:MOTOR	CF:BLOAD	D0:BSAVE
D1:DSKO$	D2:SET	D3:NAME
D4:KILL	D5:IPL	D6:COPY
D7:CMD	D8:LOCATE	D9:TO
DA:THEN	DB:TAB(	DC:STEP
DD:USR	DE:FN	DF:SPC(
E0:NOT	E1:ERL	E2:ERR
E3:STRING$	E4:USING	E5:INSTR
E6:'	E7:VARPTR	E8:CSRLIN
E9:ATTR$	EA:DSKI$	EB:OFF

```
EC:INKEY$      ED:POINT       EE:>
EF:=           F0:<           F1:+
F2:-           F3:*           F4:/
F5:^           F6:AND         F7:OR
F8:XOR         F9:EQV         FA:IMP
FB:MOD         FC:\           FD:<NVT>
FE:<NVT>       FF:<NVT>
```

In bovenstaande tabel zijn de waarden van de tokens hexadecimaal opgenomen. In deel 1 van deze serie werd reeds een programma opgenomen om deze waarden naar het decimale stelsel om te rekenen.

7.2 KONSTANTEN

Wanneer we bijvoorbeeld de programmaregel:

```
10 LET A=12
```

intoetsen, dan kennen we in het programma de konstante 12 toe aan de variabele A.

Konstanten zijn er binnen MSX in vele soorten. Steeds kiest het MSX-basic een vorm van konstante waarbij het geheugenbeslag minimaal blijft.

De volgende konstanten zijn binnen MSX-basic mogelijk:

1) *de unsigned integer konstante*

De unsigned integer konstante is een konstante die minimaal gelijk is aan 0 en maximaal gelijk is aan 65535. Hij legt beslag op drie bytes van het computergeheugen.

Het eerste byte van deze konstante kan de waarde 0D of 0E hexadecimaal bevatten. Dit byte is het voorloop-byte en geeft aan dat de volgende twee bytes een unsigned integer konstante bevatten.

De twee opvolgende bytes bevatten de konstante in binaire vorm genoteerd.

De konstante die wordt voorafgegaan door het voorbyte 0D heeft een speciale betekenis. Tijdens het 'runnen' van een basic programma

worden alle sprongadressen, allemaal 0E-konstanten, vervangen door 0D-konstanten. In deze konstanten wordt dan het direkte geheugenadres opgenomen van de regel waarnaar wordt gesprongen.

Wanneer we programmeren:

```
100 GOTO 500
```

dan zal de konstante 500 in dit voorbeeld een konstante met voorbyte 0C zijn. Tijdens executie van het programma wordt deze konstante door een 0D-type konstante vervangen. In deze konstante wordt dan het direkte geheugenadres opgenomen. Alhoewel de basic-programmeur daar niets van merkt in zijn programmaregels, resulteert dit in het feit dat een programma na korte tijd geleidelijk sneller gaat werken.

2) *de single byte integer konstante*

Deze konstante wordt voorafgegaan door het voorbyte 0F en beslaat in totaal twee bytes. Een singel byte konstante kan slechts de waarden 0 t/m 255 bevatten en wordt binair gekodeerd.

3) *de single digit integer konstante*

Deze konstante heeft geen voorbyte en kan slechts de waarde 0...9 bevatten. Hexadecimaal corresponderen de waarden 11 t/m 1A met de cijfers 0 t/m 9.

4) *de signed integer konstante*

Deze konstante wordt voorafgegaan door het voorbyte 1C en kan de waarden –32768 t/m 32767 bevatten. Na het voorbyte volgen twee bytes waarin de konstante in 2-complementnotatie is opgenomen.

5) *de single precision konstante*

Deze konstante wordt voorafgegaan door het voorbyte 1D. Na het voorbyte volgt een byte waarin het teken van de mantisse en de waarde van de exponent zijn opgenomen. Het eerste bit van dit byte is gelijk aan 0 wanneer de konstante positief is en gelijk aan 1 wanneer de konstante negatief is. De overige zeven bits vormen de waarde van de exponent (de macht van tien waarmee de mantisse moet worden vermenigvuldigd om de juiste waarde te verkrijgen). De binaire waarde van deze bits, verminderd met 65, geeft de waarde van deze exponent.

De volgende drie bytes geeft in BCD (Binary Coded Decimal, een notatie waarin per halve byte (nibble) een cijfer is gekodeerd) de zes cijfers van de mantisse aan.

6) *de double precision konstante*

Deze konstante wordt voorafgegaan door het voorbyte 1F en is qua opbouw gelijk aan de single precision konstante met dit verschil, dat de mantisse nu 7 bytes groot is.

De doorgewinterde MSX-er herkent in 4), 5) en 6) de opslagmethode die ook voor MSX-variabelen wordt gebruikt.

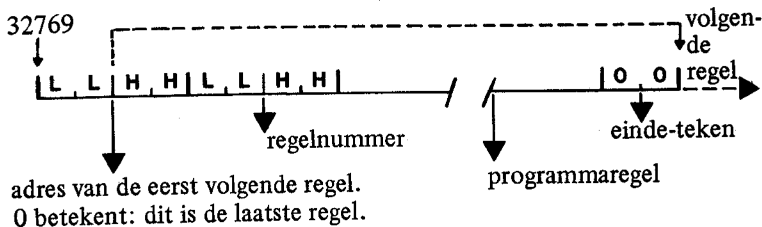
7.3 Opbouw van het basic-geheugen

Een basic-programma is op een bepaalde wijze gekodeerd in het MSX-geheugen. We zagen reeds dat de sleutelwoorden in dit geheugen vervangen werden door TOKENS. Ook zagen we hoe konstanten worden opgeslagen.

De algemene opbouw van een basic-programma in het geheugen kan men zich als volgt voorstellen:

Op adres 32768, het eerste bruikbare byte binnen MSX-basic, bevindt zich de waarde 00. Amateurs die op band of schijf (met bijvoorbeeld het programma BYTES dat in het MSX DISK HANDBOEK voor iedereen werd gepubliceerd, of met het DSP-programma dat in dit boek staat) weleens een basic-programma hebben bestudeerd, zullen weten dat op deze plaats in de programmafile juist de hexadecimale waarde FF staat. Aan deze waarde herkent MSX een programma-file.

Op adres 32769 begint pas het werkelijke basic-programma. In het volgende schema wordt uit de doeken gedaan hoe het basic-programma per regel in het computergeheugen is gekodeerd:



Zowel het adres van de volgende regel als het regelnummer zijn in LLHH-vorm gekodeerd. Dit betekent dat het eerste byte het minst significant is. Het tweede byte dient eerst met 256 te worden vermenigvuldigd waarna het eerste byte hierbij kan worden opgeteld.

Een voorbeeld:

```
PRINT PEEK (32771)+256*PEEK(32772)
```

Geeft het eerste regelnummer van het op dat moment gekodeerde programma.

Binnen de programmaregel onderkennen we TOKENS, KONSTANTEN en overige gegevens. Deze overige gegevens kunnen bestaan uit kommentaar, alfanumerieke konstanten (tussen aanhalingstekens), DATA-gegevens en variabele-namen.

Met deze kennis geladen, kunnen we vele leuke foefjes gaan uitproberen. Wat gebeurt er bijvoorbeeld wanneer ik alle regelnummers gelijk aan nul maak? Wat gebeurt er wanneer ik op geheugenlokatie 32768 een andere waarde dan 00 plaats? Wat gebeurt er wanneer ik de adressen van de eerstvolgende regels vermink?

Het hoofdstuk 'beveiliging' dat eerder in dit boek werd opgenomen, wordt plotseling een stuk duidelijker wanneer je de opbouw van het basic-geheugen een beetje kent. Door bijvoorbeeld meteen het eerste regelnummer op 65535 te stellen, maak je de rest van het programma onleesbaar. Het werkt alleen ook niet meer goed en hierop moet dan weer een andere truuk worden bedacht.

Eén van de dingen die men kan doen wanneer de opbouw van het basic-geheugen bekend is, is het onderzoeken van het variabelengebruik:

7.4 Cross reference programma

Vooral bij wat grotere programma's ontstaat vaak de vraag: „Waar heb ik welke variabele ook al weer gebruikt?" Wanneer men overweegt, een nieuwe variabele te gaan gebruiken, is het belangrijk om te weten dat deze niet elders in het programma wordt gebruikt.

Het onderstaande programma kan bij elk bestaand programma worden geMERGEd. Dit programma mag dan natuurlijk geen regelnummers groter dan 59999 hebben.

Dit programma onderzoekt het programma waarbij het is geplaatst en geeft uiteindelijk een overzicht van de gebruikte variabelen en in welke regels deze variabelen worden gebruikt.

Na dit onderzoek is het variabelen-gebruik glashelder. Start het onderzoek, nadat deze routine is geMERGEd, op met een RUN 60000. Vooral voor grotere programma's heeft deze routine wel even de tijd nodig.

De amateur-met-printer heeft deze routine natuurlijk in een mum van tijd omgebouwd tot een routine die de output op de printer geeft.

Wanneer u behalve dat u het onderstaande programma gebruikt, het programma ook bestudeert, zult u zien dat de routine het bovenliggende basic-programma onderzoekt t/m regelnummer 59999. Elke regel wordt op TOKENS en konstanten gekontroleerd. Uiteindelijk worden de variabele-namen uit de programmaregels gefilterd en geregistreerd. Dit registreren gebeurt onmiddellijk op volgorde van alfabeth.

Vòor de zeer diepgaande amateur: merk op dat voor het FN-token, het REM-token, het ERASE-token en het DATA-token een speciale programmagang bestaat. Het AS-token bestaat niet. Als zodanig is de variabele AS, wanneer in gebruik, nooit vertegenwoordigd in de cross reference.

```
60000 SCREEN 0:WIDTH 38:KEY OFF:COLOR 15
,4,4:CLS:CLEAR 16384:PP=0:DIM SS$(200):P
O=32773!:AH=0:ER=0:NF=0:DA=0
60010 PRINT TAB(6);"CROSS REFERENCE PROG
RAMMA":PRINT STRING$(37,"-"):PRINT " VAR
  REGEL REGEL REGEL REGEL REGEL":PRINT S
TRING$(37,"-")
60020 RE=PEEK(PO-1)*256+PEEK(PO-2):IF RE
>59999! THEN 60190 ELSE LOCATE 0,5,0:PRI
NT USING "BEZIG MET REGEL: #####      ";
RE
60030 IN=PEEK(PO):PO=PO+1:IF IN=0 THEN A
H=0:DA=0:NF=0:ER=0:PO=PO+4:GOTO 60020 EL
SE IF IN=34 THEN AH=-(AH=0):GOTO 60030
60040 IF AH+DA+NF THEN GOTO 60070
60050 IF IN>64 AND IN<91 THEN 60090
```

```
60060 IF IN=58 THEN ER=0:NF=0:NF=0 ELSE
IF IN=165 THEN ER=1 ELSE IF IN=40 THEN N
F=0
60070 IF IN=222 THEN NF=1 ELSE IF IN=132
 OR IN=143 THEN DA=1
60080 PO=PO-(IN=15)-2*(IN=13 OR IN=14 OR
 IN=28)-4*(IN=29)-8*(IN=31):GOTO 60030
60090 V$=CHR$(IN)
60100 IN=PEEK(PO):IF (IN>47 AND IN<58) O
R (IN>64 AND IN<91) OR (IN>32 AND IN<38)
 THEN V$=V$+CHR$(IN):PO=PO+1:GOTO 60100
60110 IF IN=40 OR ER THEN V$=V$+"()"
60120 IF V$="AS" THEN 60030
60130 IF LEN(V$)>2 THEN IN=ASC(MID$(V$,3
,1)):IF (IN>47 AND IN<58) OR (IN>64 AND
IN<91) THEN V$=LEFT$(V$,2)+MID$(V$,4):GO
TO 60130
60140 V$=LEFT$(V$+"     ",5):PL$=CHR$(RE
/256)+CHR$(RE-256*INT(RE/256)):LOCATE 23
,5:PRINT V$:FOR PL=0 TO PP
60150 IF PL=PP THEN SS$(PP)=V$+PL$:NEXT
PL:PP=PP+1:GOTO 60030
60160 IF LEFT$(SS$(PL),5)<V$ THEN NEXT P
L ELSE IF V$=LEFT$(SS$(PL),5) THEN IF PL
$>RIGHT$(SS$(PL),LEN(PL$)) THEN SS$(PL)=
SS$(PL)+PL$:GOTO 60030 ELSE 60030
60170 PP=PP+1:FOR PS=PP TO PL+1 STEP -1:
SS$(PS)=SS$(PS-1):NEXT PS:SS$(PS)=V$+PL$
60180 GOTO 60030
60190 LOCATE 0,5:PRINT SPACE$(32):LOCATE
 0,4:IF PP=0 THEN 60240 ELSE FOR IN=0 TO
 PP-1:PRINT LEFT$(SS$(IN),5);
60200 FOR PS=6 TO LEN(SS$(IN)) STEP 2:IF
 PS>6 AND ((PS-6)/2) MOD 5=0 THEN PRINT:
PRINT TAB(5);
60210 RE=ASC(MID$(SS$(IN),PS))*256+ASC(M
```

```
ID$(SS$(IN),PS+1)):PRINT USING " #####";
RE;:NEXT PS:PRINT:PRINT STRING$(37,"-")
60220 LOCATE ,,1:PRINT"(RETURN)";
60230 IF INKEY$="" THEN 60230 ELSE LOCAT
E 0,CSRLIN:NEXT IN
60240 STOP
```

Als voorbeeld volgt hier de cross-reference van het programma Sound Register Editor, dat eerder in dit boek werd geplaatst:

CROSS REFERENCE PROGRAMMA

VAR	REGEL	REGEL	REGEL	REGEL	REGEL
A$	120	160	170	180	320
	330	340	520	650	
ED	410	500			
EF	300	310	480		
F1	350	360	390	490	630
F2	350	370	390	490	630
F3	350	380	390	490	630
G	430	440	450	500	
G0	430	510	560		
G1	430	510	560		
G2	440	510	570		
G3	440	510	570		

G4	450	510	580		
G5	450	510	580		
G6	460	510	590		
G7	470	510	640		
G8	490	510	600		
G9	490	510	610		
GA	490	510	620		
GB	500	510	630		
GC	500	510	630		
GD	480	510	630		
R1	190	320	350	360	590
	600				
R2	190	330	350	370	590
	610				
R3	190	340	350	380	590
	620				
T1	90	190	320	350	360
	430	560	600		
T2	100	190	330	350	370
	440	570	610		

T3	110	190	340	350	380
	450	580	620		
TR	150	460			
V1	360	490			
V2	370	490			
V3	380	490			

8 Professioneel programmeren

8.1 Verbeteringen

In deel 1 van deze serie nam ik een tweetal programma's op die bij nader inzien nog wat verder konden worden uitgebreid.

Het eerste programma betreft de ingaveroutine. De onderstaande versie van deze ingaveroutine bevat de mogelijkheid om ook een perfekte numerieke ingave te besturen.

De routine dient te worden aangesproken nadat een WIDTH 40 is uitgevoerd en nadat de variabele I$ als volgt is ingevuld:

```
I$="XXYYTTTOA---A"
```

XX=horizontale ingavepositie
YY=vertikale ingavepositie
TTT=aantal tekens
0=indikatie ingave alfanumeriek
A---A=voorlooptekst (voor ingave)

of:

```
I$="XXYYVVN1A---A"
```

XX=horizontale ingavepositie
YY=vertikale ingavepositie
VV=aantal toegestane voorkomma-posities
N=aantal toegestane nakomma-posities
I=indikatie numerieke ingave
A---A=voorlooptekst (voor ingave)

Nadat I$ op juiste wijze is gevuld, kan een GOSUB 10000 plaatsvinden. Na terugkeer uit deze subroutine is II$ gevuld met de ingave. Deze

ingave voldoet aan de in I$ opgegeven specificaties.

Voorafgaand aan de GOSUB 10000 kan de variabele IO$ met de 'oude waarde' van de ingave worden gevuld. De ingave wordt dan met deze waarde ingevuld waarna deze kan worden veranderd of met een RETURN-toets kan worden overgenomen.

Twee voorbeelden:

1) *een ingave van een naam van maximaal 24 posities:*

```
100 I$="00100240NAAM":GOSUB 10000
```

2) *een ingave van een saldo van –999.99/9999.99 minimaal/maximaal:*

```
200 I$="00150421SALDO";GOSUB 10000
```

Wanneer bij ingave de rechter rand van het scherm wordt benaderd, dan werkt deze routine als een soort 'lichtkrant'. De opbouw van het scherm wordt niet aangetast maar het ingave-veld wordt bij elk ingegeven teken een positie naar links verschoven. Zo kan bij ingewikkelde programma's het scherm intensief worden gebruikt.

De volgende routine is wat aan de lange kant maar laat zich in een professioneel programma uitstekend gebruiken. Hij kan zelfs de basis vormen voor een in basic geschreven database-programma.

Het belangrijkste aspekt van deze routine is, dat een programma hiermee 'fool-proof' kan worden gemaakt. De ingave is zo beschermd dat zelfs een totaal onkundige gebruiker er geen rommeltje van kan maken. Prima voor programmatuur die later door anderen moet kunnen worden gebruikt.
Merk op dat deze routine voorziet in een *knipperende cursor* tijdens het ingeven!

```
10 REM ****************************
20 REM *    INGAVEROUTINE         *
30 REM *    -------------         *
40 REM * I$="XXYYTTTOA---A"       *
50 REM * OF                       *
60 REM * I$="XXYYVVN1A---A"       *
70 REM *                          *
```

```
80 REM *      XX=HOR. POSITIE      *
90 REM *      YY=VERT. POSITIE     *
100 REM*   TTTO=AANTAL TEKENS      *
110 REM*   VVN1=VOOR/NAKOMMAPOS    *
120 REM*  A---A=INPUT-TEKST        *
130 REM*                           *
140 REM* II$=INGAVERESULTAAT       *
150 REM* IO$=OUDE WAARDE           *
160 REM*                           *
170 REM* TOETSEN:                  *
180 REM* HOME   =WISSEN INGAVE     *
190 REM* DEL    =TEKEN WISSEN      *
200 REM* INS    =TEKEN INVOEGEN    *
210 REM* RETURN =EINDE INGAVE      *
220 REM* +/-    =TEKEN EN EVT.     *
230 REM*         EINDE INGAVE      *
240 REM*         (NUMERIEK)        *
250 REM*                           *
260 REM* PIJLEN:                   *
270 REM* LINKS  =CURSOR LINKS      *
280 REM* RECHTS =CURSOR RECHTS     *
290 REM* OMHOOG =TEKEN INVOEGEN    *
300 REM* OMLAAG =TEKEN WISSEN      *
310 REM*                           *
320 REM****************************
330 REM
60000 XX=VAL(LEFT$(I$,2)):YY=VAL(MID$(I$
,3,2)):TT=VAL(MID$(I$,5,3)):SS=VAL(MID$(
I$,8,1)):VV=INT(TT/10):NN=TT-10*VV:II$=I
O$:IP=LEN(II$):IP=IP-(IP=0):PG=0
60010 IF SS THEN TT=VV+NN-(NN>0)
60020 II$=II$+SPACE$(TT-LEN(II$)):PG=0
60030 LOCATE XX,YY: PRINT MID$(I$,9);":"
;LEFT$(II$,47-LEN(I$)-XX);
60040 X1=XX+LEN(I$)+IP-8:IF X1>38 THEN L
OCATE XX+LEN(I$)-7,YY:PRINT MID$(II$,X1-
```

```
38,40-POS(0)):X1=39
60050 LOCATE X1,YY,LK:LK=-(LK=0)
60060 IF EK=10 THEN EK=0:GOTO 60050 ELSE
 EK=EK+1:KK$=INKEY$:IF KK$="" THEN 60060
 ELSE KK=ASC(KK$)
60070 IF KK=28 THEN IP=IP+1+(IP=LEN(II$)
):GOTO 60040
60080 IF KK=29 THEN IP=IP-1-(IP=1):GOTO
60040
60090 IF KK=30 OR KK=18 THEN II$=LEFT$(I
I$,IP-1)+" "+MID$(II$,IP,LEN(II$)-IP):PR
INT MID$(II$,IP,40-X1);:GOTO 60040
60100 IF KK=31 OR KK=127 THEN IF IP<TT T
HEN II$=LEFT$(II$,IP-1)+MID$(II$,IP+1)+"
 ":PRINT MID$(II$,IP,40-X1);:GOTO 60040
ELSE 60040
60110 IF KK=11 THEN IO$="":GOTO 10
60120 IF KK<>13 THEN 60140 ELSE IF SS=0
THEN RETURN ELSE 60210
60130 RETURN
60140 IF SS=0 THEN 60180 ELSE IF (KK<48
OR KK>57) AND KK<>43 AND KK<>45 AND KK<>
46 THEN KK=0:GOTO 60180
60150 IF (INSTR(II$,".") AND KK=46 AND M
ID$(II$,IP,1)<>".") OR ((INSTR(II$,"-")
OR INSTR(II$,"+")) AND MID$(II$,IP,1)<>"
-" AND MID$(II$,IP,1)<>"+" AND (KK=43 OR
 KK=45)) THEN KK=0:GOTO 60180
60160 IF II$>SPACE$(LEN(II$)) AND (KK=43
 OR KK=45) AND MID$(II$,IP,1)<>"+" AND M
ID$(II$,IP,1)<>"-" THEN II$=KK$+II$:GOTO
 60210
60170 GOTO60190
60180 IF KK<32 OR KK>126 THEN BEEP:GOTO
60060
60190 MID$(II$,IP,1)=KK$:PRINT KK$;:IP=I
```

```
P+1:IF IP>TT THEN IP=TT
60200 GOTO 60040
60210 II$=STR$(VAL(II$)):IF LEFT$(II$,1)
=" " THEN II$=MID$(II$,2)
60220 IF LEFT$(II$,1)="." THEN II$="0"+I
I$
60230 IF INSTR(II$+".",".")>VV+1 THEN GO
TO 10
60240 IF NN AND INSTR(II$,".")=0 THEN II
$=II$+"."
60250 IF NN AND INSTR(II$,".")<LEN(II$)-
NN THEN GOTO 10
60260 IF NN AND INSTR(II$,".")>LEN(II$)-
NN THEN II$=II$+"0":GOTO 60260
60270 IF INSTR(II$+".",".")<VV+1 THEN II
$=" "+II$:GOTO 60270
60280 IF NN=0 THEN II$=LEFT$(II$,VV)
60290 LOCATE XX,YY: PRINT MID$(I$,9);":"
;LEFT$(II$,47-LEN(I$)-XX);
60300 RETURN
```

Natuurlijk behoeven de REM-regels niet te worden ingetikt; zij verschaffen slechts een toelichting.

8.2 Alfanumeriek en meerdere kleuren

Met de MSX-computer kunnen we in 16 verschillende kleuren werken. Vaak komen deze kleuren in tekst-informatie goed tot hun recht wanneer we er wat woorden of zinsdelen willen laten uitspringen.

Helaas. In de SCREEN 0 mode kan men met de MSX niet in meerdere kleuren werken. En ook de SCREEN 1 mode biedt veel te veel beperkingen om een beetje behoorlijk met meerdere kleuren te kunnen werken.

Om toch te kunnen werken met meerdere kleuren binnen teksten, ontwierp ik de volgende subroutine; een uitbreiding op een routine die in deel 1 werd geplaatst.

Met deze routine kunnen zelfs 42 karakters op één regel worden ge-

plaatst. De te plaatsen tekst kan in elke kleur worden opgenomen.

Voorafgaand aan het gebruik van deze routine dient natuurlijk een SCREEN 2 te zijn uitgevoerd. Wanneer we dan een tekst willen plaatsen, dienen we de variabele I$ als volgt te vullen:

```
I$="XXYYCCA---A)
```

XX=horizontale printpositie
YY=vertikale printpositie
CC=kleurkode (00/15)
A---A=de af te drukken tekst

Wanneer we I$ op de juiste wijze hebben gevuld, is een GOSUB 10000 voldoende om de aangegeven tekst in de juiste kleur te doen verschijnen. Een voorbeeld:

Plaats de tekst "MEERDERE KLEUREN" op positie 12,13 op het scherm. De kleurkode is 2 (groen):

```
200 I$="121302MEERDERE KLEUREN": GOSUB
    10000
```

```
10 REM ***************************
20 REM * 42 TEKENS PER REGEL IN *
30 REM *     MEERDERE KLEUREN   *
40 REM *                        *
50 REM * I$="XXYYCCA---A"       *
60 REM * XX,YY=POSITIE HOR/VERT *
70 REM * CC    =KLEUR 00/15     *
80 REM * A---A=TEKST            *
90 REM * ZORG VOOR SCREEN 2 EN  *
100 REM* DOE EEN GOSUB 10000    *
110 REM* VOOR DE PROJEKTIE.     *
120 REM***************************
130 REM
10000 X=VAL(LEFT$(I$,2)):Y=VAL(MID$(I$,3
,2)):C=VAL(MID$(I$,5,2))
10010 COLOR C:OPEN "GRP:" AS 1:FOR I=7 T
O LEN(I$)
10020 PRESET (6*(X+I-5),8*Y):PRINT #1,MI
D$(I$,I,1):NEXT I:CLOSE #1:RETURN
```

8.3 Geheugenproblemen

MSX-basic stelt ons in staat om ongeveer 28 kilobytes te vullen met programmaregels en gegevens. En alhoewel deze ruimte best erg groot is, zult u als gevorderde amateur op sommige momenten toch in 'geheugennood' komen. Een paar grote tabellen en het geheugen zit vaak al bijna vol!

Wanneer u in geheugenproblemen komt, zit er maar één ding op: optimaliseren.

Het programma dat u schreef, kan bijvoorbeeld al behoorlijk veel kleiner worden gemaakt door er alle spaties uit te halen. Wanneer we dan ook nog alle REM-statements er uit halen, wordt het programma nog veel korter en de hoeveelheid beschikbaar geheugen weer wat groter.

Het verwijderen van de spaties uit een programma is vaak een hele klus. Het verwijderen van REM-regels is zo mogelijk een nog grotere klus omdat er vaak, bijvoorbeeld met GOTO, naar zo'n REM-regel wordt gesprongen. Ook de regel waarin de sprong werd gemaakt, moet dus worden veranderd.

Het volgende programma heet SHRINKER. Het maakt uw programma's allemaal 'een kopje kleiner' door er alle overtollige kodering uit te verwijderen.

SHRINKER verwijdert alle overtollige spaties
verwijdert alle kommentaar
hernummert uw programma volledig
optimaliseert hier en daar wat kodes

Om dit programma te kunnen toepassen, dient u eerst het te SHRINKEN programma op tape te zetten met de SAVE-opdracht (gebruik geen CSAVE). Floppy-bezitters moeten het programma met de ,A-optie saven (ASCII-save).

Nadat het programma is opgestart, vraagt de computer u om de naam van de programma-file. Hierna voert de computer drie fasen uit:

fase 1: het programma wordt ingelezen, spaties en kommentaar worden verwijderd, regels worden hernummerd en een verwijzingstabel wordt klaar gemaakt.

fase 2: de sprongadressen worden in het programma aangepast. Het programma wordt een beetje geoptimaliseerd.

fase 3: het programma wordt op band/schijf teruggeschreven.

Cassetterecorder-bezitters dienen tijdens fase 2 de recorder op opnemen te zetten. De opname zal uiteindelijk het ingekrompen maar feilloos werkende programma bevatten.

```
10 REM ***************************
20 REM *   PROGRAMMA SHRINKER    *
30 REM *   -----------------     *
40 REM *  (C) 1985 STARK TEXEL   *
50 REM ***************************
60 REM
70 REM ***************************
80 REM *      INITIALISATIE      *
90 REM ***************************
100 REM
110 MAXFILES=1:KEY OFF:SCREEN 0:WIDTH 37
:KEY OFF:COLOR 15,4,4:CLS:PRINT "PROGRAM
MA SHRINKER":PRINT
120 CLEAR 13000:DIM A$(1000),A!(1000),P(
12,1)
130 LINE INPUT "FILE :";F$
140 REM
150 REM ***************************
160 REM * FASE 1, INLEZEN PRO-    *
170 REM * GRAMMA UIT FILE F$ IN   *
180 REM * A$(), SPATIELOOS MAKEN  *
190 REM * REM'S VERWIJDEREN, RE-  *
200 REM * GELTABEL SAMENSTELLEN   *
210 REM * [A!()] EN REGELS HER-   *
220 REM * NUMMEREN                *
230 REM ***************************
240 REM
```

```
250 OPEN F$ FOR INPUT AS 1
260 FOR I=0 TO 750:IF EOF(1)=0 THEN LINE
 INPUT #1,A$(I):LOCATE 0,4:PRINT USING "
FASE 1: REGEL #####";VAL(A$(I)):LOCATE 0
,6:PRINT A$(I);SPACE$(255) ELSE CLOSE:GO
TO 560
270 P=1
280 AH=INSTR(P,A$(I),CHR$(34)):SP=INSTR(
P,A$(I)," "):IF AH=0 THEN AH=256
290 IF SP=0 THEN 320
300 IF SP<AH THEN A$(I)=LEFT$(A$(I),SP-1
)+MID$(A$(I),SP+1):GOTO 280
310 P=INSTR(AH+1,A$(I),CHR$(34)):IF P=0
THEN 320 ELSE P=P+1:GOTO 280
320 P=1:IF RIGHT$(A$(I),1)=CHR$(34) AND
MID$(A$(I),LEN(A$(I))-1,1)>" " THEN A$(I
)=LEFT$(A$(I),LEN(A$(I))-1)
330 IF RIGHT$(A$(I),1)=" " THEN A$(I)=LE
FT$(A$(I),LEN(A$(I))-1):GOTO 320
340 AH=INSTR(P,A$(I),CHR$(34)):SP=INSTR(
P,A$(I),"'"):IF AH=0 THEN AH=256
350 IF SP=0 THEN 380
360 IF SP<AH THEN A$(I)=LEFT$(A$(I),SP-1
):GOTO 380
370 P=INSTR(AH+1,A$(I),CHR$(34)):IF P=0
THEN 380 ELSE P=P+1:GOTO 340
380 P=1
390 AH=INSTR(P,A$(I),CHR$(34)):SP=INSTR(
P,A$(I),":REM"):IF AH=0 THEN AH=256
400 IF SP=0 THEN 430
410 IF SP<AH THEN A$(I)=LEFT$(A$(I),SP-1
):GOTO 430
420 P=INSTR(AH+1,A$(I),CHR$(34)):IF P=0
THEN 430 ELSE P=P+1:GOTO 390
430 P=1
440 IF MID$(A$(I),P,1)>="0" AND MID$(A$(
```

```
I),P,1)<="9" THEN P=P+1:GOTO 440
450 IF MID$(A$(I),P,3)="REM" THEN A$(I)=
""
460 ST=VAL(A$):IF LEN(STR$(ST))-1=LEN(A$
(I)) THEN A$(I)=""
470 IF A$(I)>"" THEN A!(RG)=VAL(A$(I)):A
$(I)=MID$(STR$(RG),2)+MID$(A$(I),P):A$(R
G)=A$(I):RG=RG+1
480 I=RG:NEXT I:RG=RG-1
490 REM
500 REM **************************
510 REM * FASE 2, REGELVERWIJ-     *
520 REM * ZINGEN CORRIGEREN AAN   *
530 REM * DE HAND VAN TABEL A!() *
540 REM **************************
550 REM
560 FOR I=0 TO RG:LOCATE 0,4:PRINT USING
 "FASE 2: REGEL: #####";VAL(A$(I)):LOCAT
E 0,6:PRINT A$(I);SPACE$(255)
570 P=INSTR(A$(I),"THENGOTO"):IF P=0 THE
N P=INSTR(A$(I),"ELSEGOTO"):IF P=0 THEN
590
580 A$(I)=LEFT$(A$(I),P+3)+MID$(A$(I),P+
8):GOTO 570
590 PZ=1
600 RESTORE 600:FOR K=0 TO 12:READ ST$:P
(K,0)=INSTR(PZ,A$(I),ST$):P(K,0)=P(K,0)-
100000!*(P(K,0)=0):P(K,1)=LEN(ST$):NEXT
K:DATA GOTO,GOSUB,THEN,ELSE,RENUM,AUTO,L
IST,DELETE,RUN,RESUME,RESTORE,ERL=,RETUR
N
610 P=100000!:FOR K=0 TO 12:IF P>P(K,0)
THEN P=P(K,0):L=P(K,1)
620 NEXT K
630 IF P=100000! THEN 690
640 P=P+L:PZ=P:IF MID$(A$(I),P,1)<"0" OR
```

```
 MID$(A$(I),P,1)>"9" THEN GOTO 600
650 ST$="":P1=PZ:A$(I)=A$(I)+":"
660 IF MID$(A$(I),PZ,1)>="0" AND MID$(A$
(I),PZ,1)<="9" THEN ST$=ST$+MID$(A$(I),P
Z,1):PZ=PZ+1:GOTO 660
670 ST=VAL(ST$):FOR J=0 TO RG:IF A!(J)<S
T THEN NEXT J
680 SQ$=MID$(STR$(J),2):A$(I)=LEFT$(A$(I
),P1-1)+SQ$+MID$(A$(I),PZ):A$(I)=LEFT$(A
$(I),LEN(A$(I))-1):PZ=PZ-LEN(ST$)+LEN(SQ
$):IF MID$(A$(I),PZ,1)="," THEN PZ=PZ+1:
GOTO 650 ELSE 600
690 NEXT I
700 REM
710 REM **************************
720 REM * FASE 3, SCHRIJVEN PRO- *
730 REM * GRAMMA VANUIT A$()     *
740 REM * NAAR FILE F$           *
750 REM **************************
760 REM
770 OPEN F$ FOR OUTPUT AS 1:FOR I=0 TO R
G:IF A$(I)>"" THEN LOCATE 0,4:PRINT USIN
G "FASE 3: REGEL: #####";VAL(A$(I)):PRIN
T #1,A$(I):LOCATE 0,6:PRINT A$(I);SPACE$
(255)
780 NEXT I:CLOSE
790 REM
800 REM **************************
810 REM * EINDE PROGRAMMA        *
820 REM **************************
830 REM
840 LOCATE 0,6:PRINT "FILE ";F$;" IS GES
HRINKED!";SPACE$(255):STOP
```

Als voorbeeld volgt hieronder hetzelfde programma; ditmaal echter na behandeling met zichzelf. Merk op dat het programma totaal onlees-

baar is geworden (een beetje beveiliging?) maar nog steeds feilloos werkt...

In een héél enkel geval zal het behandelde programma op een enkele plaats moeten worden gecorrigeerd. Zo ook dit voorbeeld. Op regel 28 dienen na behandeling de woorden THEN en ELSE te worden terugveranderd in THENGOTO en ELSEGOTO. In de praktijk komen dit soort situaties zelden of nooit voor.

```
0  MAXFILES=1:KEYOFF:SCREEN0:WIDTH37:KEY
OFF:COLOR15,4,4:CLS:PRINT"PROGRAMMA SHRI
NKER":PRINT
1 CLEAR13000:DIMA$(1000),A!(1000),P(12,1
)
2 LINEINPUT"FILE :";F$
3 OPENF$FORINPUTAS1
4 FORI=0TO750:IFEOF(1)=0THENLINEINPUT#1,
A$(I):LOCATE0,4:PRINTUSING"FASE 1: REGEL
 #####";VAL(A$(I)):LOCATE0,6:PRINTA$(I);
SPACE$(255)ELSECLOSE:GOTO27
5 P=1
6 AH=INSTR(P,A$(I),CHR$(34)):SP=INSTR(P,
A$(I)," "):IFAH=0THENAH=256
7 IFSP=0THEN10
8 IFSP<AHTHENA$(I)=LEFT$(A$(I),SP-1)+MID
$(A$(I),SP+1):GOTO6
9 P=INSTR(AH+1,A$(I),CHR$(34)):IFP=0THEN
10ELSEP=P+1:GOTO6
10 P=1:IFRIGHT$(A$(I),1)=CHR$(34)ANDMID$
(A$(I),LEN(A$(I))-1,1)>" "THENA$(I)=LEFT
$(A$(I),LEN(A$(I))-1)
11 IFRIGHT$(A$(I),1)=" "THENA$(I)=LEFT$(
A$(I),LEN(A$(I))-1):GOTO10
12 AH=INSTR(P,A$(I),CHR$(34)):SP=INSTR(P
,A$(I),"'"):IFAH=0THENAH=256
13 IFSP=0THEN16
14 IFSP<AHTHENA$(I)=LEFT$(A$(I),SP-1):GO
```

```
TO16
15 P=INSTR(AH+1,A$(I),CHR$(34)):IFP=0THE
N16ELSEP=P+1:GOTO12
16 P=1
17 AH=INSTR(P,A$(I),CHR$(34)):SP=INSTR(P
,A$(I),":REM"):IFAH=0THENAH=256
18 IFSP=0THEN21
19 IFSP<AHTHENA$(I)=LEFT$(A$(I),SP-1):GO
TO21
20 P=INSTR(AH+1,A$(I),CHR$(34)):IFP=0THE
N21ELSEP=P+1:GOTO17
21 P=1
22 IFMID$(A$(I),P,1)>="0"ANDMID$(A$(I),P
,1)<="9"THENP=P+1:GOTO22
23 IFMID$(A$(I),P,3)="REM"THENA$(I)="
24 ST=VAL(A$):IFLEN(STR$(ST))-1=LEN(A$(I
))THENA$(I)="
25 IFA$(I)>""THENA!(RG)=VAL(A$(I)):A$(I)
=MID$(STR$(RG),2)+MID$(A$(I),P):A$(RG)=A
$(I):RG=RG+1
26 I=RG:NEXTI:RG=RG-1
27 FORI=0TORG:LOCATE0,4:PRINTUSING"FASE
2: REGEL: #####";VAL(A$(I)):LOCATE0,6:PR
INTA$(I);SPACE$(255)
28 P=INSTR(A$(I),"THENGOTO"):IFP=0THENP=
INSTR(A$(I),"ELSEGOTO"):IFP=0THEN30
29 A$(I)=LEFT$(A$(I),P+3)+MID$(A$(I),P+8
):GOTO28
30 PZ=1
31 RESTORE31:FORK=0TO12:READST$:P(K,0)=I
NSTR(PZ,A$(I),ST$):P(K,0)=P(K,0)-100000!
*(P(K,0)=0):P(K,1)=LEN(ST$):NEXTK:DATAGO
TO,GOSUB,THEN,ELSE,RENUM,AUTO,LIST,DELET
E,RUN,RESUME,RESTORE,ERL=,RETURN
32 P=100000!:FORK=0TO12:IFP>P(K,0)THENP=
P(K,0):L=P(K,1)
```

```
33 NEXTK
34 IFP=100000!THEN40
35 P=P+L:PZ=P:IFMID$(A$(I),P,1)<"0"ORMID
$(A$(I),P,1)>"9"THEN31
36 ST$="":P1=PZ:A$(I)=A$(I)+":
37 IFMID$(A$(I),PZ,1)>="0"ANDMID$(A$(I),
PZ,1)<="9"THENST$=ST$+MID$(A$(I),PZ,1):P
Z=PZ+1:GOTO37
38 ST=VAL(ST$):FORJ=0TORG:IFA!(J)<STTHEN
NEXTJ
39 SQ$=MID$(STR$(J),2):A$(I)=LEFT$(A$(I)
,P1-1)+SQ$+MID$(A$(I),PZ):A$(I)=LEFT$(A$
(I),LEN(A$(I))-1):PZ=PZ-LEN(ST$)+LEN(SQ$
):IFMID$(A$(I),PZ,1)=","THENPZ=PZ+1:GOTO
36ELSE31
40 NEXTI
41 OPENF$FOROUTPUTAS1:FORI=0TORG:IFA$(I)
>""THENLOCATE0,4:PRINTUSING"FASE 3: REGE
L: #####";VAL(A$(I)):PRINT#1,A$(I):LOCAT
E0,6:PRINTA$(I);SPACE$(255)
42 NEXTI:CLOSE
43 LOCATE0,6:PRINT"FILE ";F$;" IS GESHRI
NKED!";SPACE$(255):STOP
```

8.4 Wacht-op-toets

Vaak komt het voor dat er in een programma-afloop gepauzeerd dient te worden. Vaak dienen de op het beeldscherm geprojekteerde gegevens eerst goed te kunnen worden gelezen voordat het computerprogramma zijn werk af maakt.

Meestal wordt zo'n pauze voorafgegaan door de tekst "GEEF RETURN" of "GEEF EEN TOETS IN" waarna het programma wacht totdat de betreffende toets is ingedrukt.

In de praktijk blijkt zo'n hele eenvoudige toepassing toch vaak een probleem te vormen. Hoe vraag ik een willekeurige toets af? Hoe kontroleer ik of alleen de RETURN werd ingegeven? Hoe vestig ik er

de aandacht op dat er een toets ingegeven dient te worden?

In het volgende voorbeeld staan vier manieren om aan dit probleem het hoofd te bieden.

Steeds wordt gevraagd om een RETURN-toets of een willekeurige toets, in het eerste geval zonder en in het tweede geval met een knipperende cursor.

Natuurlijk is het de bedoeling dat u de gewenste programmaregels op de juiste plaats in uw programma overneemt; het programma zoals het hieronder staat heeft op zichzelf natuurlijk weinig zin:

```
10 REM **************************
20 REM * WACHT OP RETURN-TOETS *
30 REM **************************
40 REM
50 LOCATE ,,1:PRINT "(GEEF RETURN)";
60 IF INKEY$<>CHR$(13) THEN 60
70 REM
80 REM **************************
90 REM* WACHT OP WILL. TOETS  *
100 REM**************************
110 REM
120 LOCATE ,,1:PRINT"(GEEF EEN TOETS IN)
";
130 IF INKEY$="" THEN 130
140 REM
150 REM**************************
160 REM* WACHT OP RETURN MET    *
170 REM* KNIPPER-CURSOR         *
180 REM**************************
190 REM
200 PRINT "(GEEF RETURN)";
210 T1=0:T2=-(T2=0)
220 IF INKEY$<>CHR$(13) THEN T1=T1+1:IF
T1=20 THEN LOCATE ,,T2:GOTO 210 ELSE 220
230 REM
```

```
240 REM**************************
250 REM* WACHT OP RETURN MET    *
260 REM* KNIPPER-CURSOR         *
270 REM**************************
280 REM
290 PRINT "(GEEF EEN TOETS IN)";
300 T1=0:T2=-(T2=0)
310 IF INKEY$="" THEN T1=T1+1:IF T1=20 T
HEN LOCATE ,,T2:GOTO 300 ELSE 310
```

9 Een beetje wiskunde

9.1 Graden en radialen

Vaak heeft men er moeite mee om een hoek, gegeven in graden, minuten en seconden, om te rekenen naar radialen. Toch zal dit met MSX-basic vaak moeten; alle goniometrische funkties verlangen de hoekmaat namelijk in radialen of verstrekken de hoekmaat in radialen.

Met de volgende funktie is het een fluitje van een cent geworden om een hoekmaat, gegeven in graden, minuten en seconden, om te rekenen naar een hoek in radialen.

Nadat het volgende programma is uitgevoerd, kan de hoekmaat R in radialen uit de graden (G), minuten (M) en seconden (S) op de volgende manier worden berekend:

```
H=FNG(G,M,S)
```

```
10 REM ******************************
20 REM * VAN GRADEN NAAR RADIALEN *
30 REM * ------------------------ *
40 REM * FNG(GRAD,MIN,SEC) GEEFT   *
50 REM * DE HOEK IN RADIALEN; DE   *
60 REM * HOEKMAAT WAARMEE MSX      *
70 REM * WERKT. BIJVOORBEELD:      *
80 REM *                           *
90 REM * SIN(FNG(44,12,13))        *
100 REM*                           *
110 REM* GEEFT DE SINUS VAN 44     *
```

```
120 REM* GRADEN, 12 MINUTEN EN 13 *
130 REM* SECONDEN.                *
140 REM*****************************
150 REM
160 DEF FNG(G,M,S)=.0174532925199433#*(G+
M/60+S/3600)
```

Andersom is het vaak belangrijk om uit een hoekmaat in radialen weer de graden, minuten en seconden te kunnen bepalen.

Nadat het volgende programma is uitgevoerd, kan de hoekmaat in graden, minuten en seconden als volgt uit de hoekmaat in radialen (R) worden berekend:

```
G$=FNR$(R)
```

Na dit kommando bevat G$ de hoek in graden (eerste vijf posities), minuten (tweede vijf posities) en seconden (derde vijf posities). Met de VAL-funktie kunnen deze waarden uit G$ verder worden bepaald.

```
10 REM ****************************
20 REM * VAN RADIALEN NAAR GRADEN *
30 REM * ------------------------ *
40 REM * FNR$(RADIALEN) GEEFT     *
50 REM * IN STRINGVORM DE HOEK    *
60 REM * IN GRADEN,MINUTEN EN SE- *
70 REM * CONDEN. (IEDER 5 POSI-   *
80 REM * TIES. BIJVOORBEELD:      *
90 REM *                          *
 100 REM* LET A$=FNR$(3.1415)     *
 110 REM*                         *
 120 REM* NA DIT BEVEL BEVAT A$:  *
 130 REM*                         *
 140 REM* " 179  59   40   "      *
 150 REM*                         *
 160 REM* 3.1415 RADIALEN=179 GRA- *
 170 REM* DEN,59 MINUTEN EN 40 SE- *
 180 REM* CONDEN.                  *
```

```
190 REM*                          *
200 REM* G=VAL(LEFT$(A$,5))       *
210 REM* M=VAL(MID$(A$,6,5))      *
220 REM* S=VAL(MID$(A$,11,5))     *
230 REM*                          *
240 REM* GEEFT IN G,M EN S DE     *
250 REM* GRADEN,MINUTEN EN SECON- *
260 REM* DEN.                     *
270 REM***************************
280 REM
290 DEF FNR(X)=57.29577951308Z#*X
300 DEF FNR1$(X)=LEFT$(STR$(X)+"      ",5
)
310 DEF FNR$(X)=FNR1$(FIX(FNR(X)))+FNR1$
(FIX((FNR(X)-FIX(FNR(X)))*60))+FNR1$(FIX
((FNR(X)*60-FIX(FNR(X)*60))*60))
```

9.2 Ontbrekende gonio-funkties

Op elke rekenmachine kennen we ze, de arcsinus en de arccosinus. Twee goniomètrische funkties die we op de MSX-computer helaas moeten ontberen.

Echter, met de volgende, user defined funkties, maken we aan deze situatie voorgoed een einde.

Nadat het volgende programma is uitgevoerd, geeft FNAS(X) de arcsinus van X en geeft FNAC(X) de arccosinus van X. Natuurlijk moeten de waarden van X in beide gevallen niet kleiner zijn dan –1 en niet groter zijn dan 1.

```
10 REM ***************************
20 REM * GONIOMETRISCHE.FUNKTIES *
30 REM * ----------------------- *
40 REM * FNAS(X) GEEFT DE ARCSI- *
50 REM * NUS VAN X IN RADIALEN.  *
60 REM * X MOET ZICH IN HET IN-  *
70 REM * TERVAL [-1;1] BEVINDEN  *
```

```
80 REM *                                  *
90 REM * FNAC(X) GEEFT DE ARCCO- *
100 REM* SINUS VAN X IN RADIA-    *
110 REM* LEN. X MOET ZICH IN HET *
120 REM* INTERVAL [-1;1] BEVIN-   *
130 REM* DEN.                     *
140 REM***************************
150 REM
160 DEF FNAS(X)=ATN(X/SQR(-X*X+1.0000000
000001#))
170 DEF FNAC(X)=(X<1)*(FNAS(X)-1.5707963
267949#)
```

10 Tenslotte

In deel 1 werden enige handige funkties gegeven teneinde datumberekeningen te kunnen doen. Ondanks onze uitnodiging stuurde niemand van de lezers ons een kalender voor 1999 op.

Om te laten zien dat je met de in deel 1 opgenomen datum-funkties een dergelijke kalender gemakkelijk kunt samenstellen, namen we het volgende programma op.

Dit programma vraagt u eerst om een jaarnummer (00=1900, 99=1999) waarna, maand voor maand, de kalender van dat jaar op beeld verschijnt, inclusief de weeknummering!

Een handige amateur past dit programma in een oogwenk aan voor zijn of haar printer.

Veel plezier met dit en alle andere programma's en tot in deel 3...

```
10 REM ***************************
20 REM *  KALENDER GENERATOR     *
30 REM ***************************
40 REM
50 DATA JANUARI,FEBRUARI,MAART,APRIL,MEI
,JUNI,JULI,AUGUSTUS,SEPTEMBER,OKTOBER,NO
VEMBER,DECEMBER
60 DEF FND1(X)=INT(X/10000)
70 DEF FND2(X)=INT(X/100)-100*FND1(X)
80 DEF FND3(X)=X-10000*FND1(X)-100*FND2(
X)
90 DEF FNK2(X)=1+365.25*FND3(X)+FND1(X)-
```

```
1+VAL(MID$("000031059090120151181212243273304334",(FND2(X)-1)*3+1,3))-(FND3(X) MOD 4 =0)*(FND2(X)<3)
100 DEF FNK1(X)=INT(FNK2(X))-7*INT(FNK2(X)/7)
110 KEY OFF:COLOR 15,4,4:WIDTH 40:DIM D(6,5)
120 INPUT "JAAR ";J:IF J<0 OR J>99 OR J>INT(J) THEN 120
130 S=FNK1(10100+M*100+J):CLS
140 W=1:RESTORE:FOR M=1 TO 12:ERASE D:DIM D(6,5)
150 MX=31:IF (M-(M>7)) MOD 2=0 THEN MX=30
160 IF M=2 THEN MX=28-(J MOD 4=0)
170 R=0:FOR I=1 TO MX:D(S,R)=I:S=S+1:IF S=7 THEN S=0:R=R+1
180 NEXT I:READ M$
190 PRINT M$;TAB(20);"19";RIGHT$(STR$(J+100),2)
200 PRINT STRING$(24,"-"):PRINT"WK  ZO MA DI WO DO VR ZA":PRINT STRING$(24,"-")
210 FOR I=0 TO 5
220 IF W=1 AND I=0 AND D(4,0)=0 THEN W=53
230 IF W=53 AND M=12 AND D(4,R)=0 THEN W=1
240 IF D(0,I)+D(6,I) THEN PRINT USING "## ";W; ELSE PRINT:GOTO 280
250 IF D(6,I) THEN W=W+1:IF W=54 THEN W=1
260 FOR K=0 TO 6:IF D(K,I) THEN PRINT USING "## ";D(K,I); ELSE PRINT "   ";
270 NEXT K:PRINT:NEXT I:PRINT
280 IF INKEY$="" THEN 280 ELSE NEXT M:RUN
```